РЕД База Данных

Вариант исполнения РС.РБД.5

Руководство по SQL

(Руководство пользователя)

46.29926343.502120-01 93 2

Содержание

Введение 22

1 SQL РЕД Базы Данных. Общие сведения 24
1.1 Структура данных на внешнем носителе (ODS) . . . 24
1.2 Объекты базы данных . . . 25
1.3 Подмножества языка . . . 27
1.4 Диалекты базы данных . . . 28
1.5 Операторы. Общие сведения . . . 29
1.6 Ключевые слова. Общие сведения . . . 29
1.7 Идентификаторы . . . 29
1.8 Константы (литералы) . . . 30
1.8.1 Строковые константы . . . 30
1.8.2 Числовые константы . . . 32
1.8.3 Константы даты и времени . . . 34
1.9 Операторы SQL . . . 34
1.9.1 Оператор конкатенации . . . 35
1.9.2 Арифметические операторы . . . 36
1.9.3 Оператор сравнения . . . 36
Оператор равенства . . . 37
1.9.4 Логические операторы . . . 37
1.10 Специальные символы . . . 39
1.11 Комментарии . . . 39

2 Выражения 40
2.1 Подзапросы . . . 41
2.1.1 Коррелированные подзапросы . . . 41
2.1.2 Подзапросы возвращающие скалярный результат . . . 42
2.2 Предикаты . . . 42
2.2.1 Предикаты сравнения . . . 42
Оператор BETWEEN . . . 42
Оператор LIKE . . . 43
Оператор STARTING WITH . . . 44
Оператор CONTAINING . . . 44
Оператор SIMILAR TO . . . 45
Оператор IS DISTINCT FROM . . . 48
Оператор IS . . . 48
2.2.2 Предикаты существования . . . 49
Предикат EXISTS . . . 49
Предикат IN . . . 49
Предикат SINGULAR . . . 50
2.2.3 Количественные предикаты подзапросов . . . 51
Оператор ALL . . . 51
Операторы SOME и ANY . . . 51

3 Типы данных РЕД Базы Данных 52
3.1 Перечень типов данных . . . 52
3.2 SMALLINT . . . 54
3.3 INTEGER . . . 55
3.4 BIGINT . . . 55
3.5 INT128 . . . 55
3.6 Шестнадцатеричный формат для целых чисел . . . 56

3.7 Восьмеричный формат для целых чисел . . . 56
3.8 Двоичный формат для целых чисел . . . 57
3.9 FLOAT . . . 57
3.10 DOUBLE PRECISION . . . 58
3.11 DECFLOAT . . . 58
3.11.1 Режимы округления . . . 59
3.11.2 Семантика сравнения . . . 60
3.11.3 Обработка ошибок . . . 61
3.11.4 Поддержка в клиентских приложениях . . . 62
3.11.5 Литералы констант DECFLOAT . . . 62
3.12 NUMERIC . . . 63
3.13 DECIMAL . . . 64
3.14 DATE . . . 65
3.15 TIME . . . 66
3.16 TIMESTAMP . . . 67
3.17 CHAR . . . 68
3.18 VARCHAR . . . 68
3.19 NCHAR . . . 68
3.20 Операции и функции для строковых данных . . . 68
3.21 Строковые литералы . . . 69
3.22 Наборы символов . . . 70
3.22.1 UNICODE . . . 70
3.22.2 Набор символов NONE и OCTETS . . . 71
3.22.3 Порядок сортировки . . . 71
Регистронезависимый поиск . . . 71
Порядок сортировки для UTF-8 . . . 71
3.22.4 Индексирование символьных типов . . . 72
3.23 BOOLEAN . . . 72
3.24 BINARY . . . 73
3.25 VARBINARY . . . 73
3.26 BLOB . . . 74
3.26.1 Хранение BLOB . . . 75
3.27 Массивы . . . 76
3.28 SQL_NULL . . . 77
3.29 Явное преобразование типов данных. Функция CAST . . . 77
3.29.1 Преобразование в типы данных даты и времени . . . 78
3.29.2 Преобразование из типов данных даты и времени в строку . . . 81
3.29.3 Преобразование из строки в типы данных даты и времени . . . 83
3.30 Неявное преобразование типов данных . . . 85

4 JSON 86
4.1 Введение в SQL/JSON . . . 86
4.2 Язык путей JSON . . . 87
4.2.1 Режимы: lax и strict . . . 87
4.3 Доступ к элементам . . . 88
4.3.1 Обращение к элементу объекта . . . 89
4.3.2 Обращение к элементу массива . . . 89
4.4 Методы элементов . . . 90
4.4.1 Метод type() . . . 90
4.4.2 Метод size() . . . 91
4.4.3 Числовые методы элементов: double(), ceiling(), floor(), abs() . . . 91
4.4.4 Метод keyvalue() . . . 92
4.5 Арифметические операции в пути . . . 93
4.5.1 Унарный плюс и минус . . . 93

4.5.2 Бинарные операции . . . 94
4.6 Фильтры . . . 94
4.6.1 Обработка ошибок в фильтрах . . . 95
4.6.2 Логические операторы . . . 95
4.6.3 Операторы сравнения . . . 96
4.6.4 Предикат like_regex . . . 96
4.6.5 Предикат starts with . . . 97
4.6.6 Предикат exists . . . 97
4.6.7 Предикат is unknown . . . 98
4.7 Функции SQL/JSON . . . 98
4.7.1 Общий синтаксис функций работы с данными . . . 98
4.7.2 Значение JSON . . . 99
4.7.3 Выражение пути . . . 99
4.7.4 Оператор передачи контекстных переменных . . . 100
4.7.5 Выходное значение . . . 100
4.8 Функции работы с данными . . . 101
4.8.1 JSON_VALUE . . . 101
4.8.2 JSON_QUERY . . . 102
4.8.3 JSON_MODIFY . . . 103
4.8.4 JSON_TABLE . . . 105
4.8.5 Вложенные столбцы . . . 108
4.8.6 Предложение PLAN . . . 108
4.9 Функции валидации . . . 111
4.9.1 JSON_EXISTS . . . 111
4.9.2 IS JSON . . . 112
4.10 Функции генерации данных . . . 114
4.10.1 Общий формат входных значений . . . 114
4.10.2 JSON_OBJECT . . . 115
4.10.3 JSON_OBJECTAGG . . . 116
4.10.4 JSON_ARRAY . . . 117
4.10.5 JSON_ARRAYAGG . . . 117

5 Транзакции 119
5.1 SET TRANSACTION . . . 119
5.1.1 Имя транзакции . . . 120
5.1.2 Параметры транзакции . . . 121
5.1.3 Режим доступа . . . 121
5.1.4 Режим разрешения блокировок . . . 121
5.1.5 Уровень изоляции . . . 122
Уровень изоляции SNAPSHOT . . . 122
Уровень изоляции SNAPSHOT TABLE STABILITY . . . 123
Уровень изоляции READ COMMITTED . . . 123
5.1.6 Опция NO AUTO UNDO . . . 125
5.1.7 Опция AUTO COMMIT . . . 125
5.1.8 Опция IGNORE LIMBO . . . 125
5.1.9 Опция AUTO RELEASE TEMP BLOBID . . . 126
5.1.10 Предложение RESERVING . . . 126
5.2 Подтверждение транзакции . . . 127
5.3 Откат (отмена) транзакции . . . 128
5.4 Использование вложенных транзакций . . . 129
5.5 Внутренние точки сохранения . . . 130
5.6 Точки сохранения и PSQL . . . 130
5.7 Вариант взаимной блокировки . . . 130

6 Базы данных (DATABASE) 132
6.1 Создание базы данных . . . 132
6.1.1 Спецификация файла . . . 133
6.1.2 Спецификация удаленного сервера . . . 133
6.1.3 USER, PASSWORD, ROLE . . . 134
6.1.4 PAGE_SIZE . . . 134
6.1.5 LENGTH . . . 134
6.1.6 SET NAMES, DEFAULT CHARACTER SET . . . 135
6.1.7 DIFFERENCE FILE . . . 135
6.1.8 Вторичные файлы . . . 135
6.2 Примеры создания базы данных . . . 136
6.3 Соединение с существующей базой данных . . . 139
6.4 Строки подключения . . . 140
6.4.1 Строка подключения к локальной базе данных . . . 140
6.4.2 Строка подключение через TCP/IP . . . 141
6.4.3 URL-подобная строка подключения . . . 141
6.5 Изменение существующей базы данных . . . 142
6.5.1 Добавление вторичного файла . . . 142
6.5.2 Изменение пути и имени дельта файла . . . 143
6.5.3 Перевод базы данных в режим «безопасного копирования» . . . 143
6.5.4 Изменение набора символов по умолчанию . . . 143
6.5.5 LINGER . . . 143
6.5.6 Изменение привилегий выполнения по умолчанию . . . 144
6.5.7 Шифрование базы данных . . . 144
6.6 Удаление базы данных . . . 144
6.7 Создание примечаний для базы данных и объектов базы данных . . . 145

7 Оперативные копии (SHADOW) 147
7.1 Создание оперативной копии . . . 147
7.1.1 Режимы AUTO и MANUAL . . . 148
7.1.2 Файл оперативной копии . . . 148
7.1.3 LENGTH . . . 149
7.1.4 Вторичные файлы . . . 149
7.2 Удаление оперативной копии . . . 149

8 Домены (DOMAIN) 151
8.1 Создание домена . . . 151
8.1.1 Задание типа данных . . . 153
8.1.2 Значение по умолчанию . . . 154
8.1.3 Значение NOT NULL . . . 155
8.1.4 Условие домена . . . 155
Ключевое слово VALUE . . . 156
8.1.5 Ключевое слово COLLATE . . . 157
8.2 Изменение домена . . . 157
8.3 Удаление домена . . . 159
8.4 Примечание домена . . . 160

9 Таблицы (TABLE) 161
9.1 Создание таблиц . . . 161
9.1.1 Глобальные временные таблицы (GTTs) . . . 162
9.1.2 Использование внешних файлов . . . 163
Файлы с фиксированной длиной строк . . . 164
Файлы формата CSV . . . 166
9.1.3 Задание типа данных . . . 167

9.1.4 Использование ссылки на домен . . . 168
9.1.5 Значение по умолчанию . . . 168
9.1.6 Значение NOT NULL . . . 169
9.1.7 Ограничения . . . 169
Именованные ограничения . . . 170
Имена для ограничений и их индексов . . . 170
Предложение USING . . . 170
Ограничение UNIQUE . . . 171
Ограничение PRIMARY KEY . . . 172
Ограничение FOREIGN KEY, REFERENCES . . . 173
Ограничение CHECK . . . 175
9.1.8 Нумерация столбцов . . . 176
9.1.9 Вычисляемые столбцы . . . 176
9.1.10 Столбцы идентификации . . . 179
9.1.11 Привилегии выполнения . . . 179
9.2 Изменение таблиц . . . 179
9.2.1 Добавление нового столбца . . . 181
9.2.2 Добавление ограничения таблицы . . . 182
9.2.3 Удаление столбца таблицы . . . 183
9.2.4 Удаление ограничения . . . 184
9.2.5 Изменение существующего столбца . . . 184
Изменение имени . . . 184
Изменение типа данных столбца . . . 185
Изменение позиции столбца . . . 185
Удаление значения по умолчанию столбца . . . 185
Добавление значения по умолчанию столбца . . . 186
Добавление ограничения NOT NULL . . . 186
Удаление ограничения NOT NULL . . . 186
Изменение вычисляемых столбцов . . . 186
Изменение столбцов идентификации . . . 187
9.2.6 Изменение прав на работу с таблицей . . . 188
9.2.7 Перемещение таблицы в табличное пространство . . . 188
9.3 Удаление таблиц . . . 188
9.4 Пересоздание таблицы . . . 189
9.5 Примечание к таблице и ее столбцам . . . 189

10 Табличное пространство (TABLESPACE) 191
10.1 Примечание для табличного пространства . . . 193

11 Операторы DML 194
11.1 SELECT . . . 194
11.1.1 WITH RECURSIVE . . . 196
11.1.2 Список выбора . . . 198
11.1.3 DISTINCT, ALL . . . 200
11.1.4 FIRST и SKIP . . . 200
11.1.5 FROM . . . 201
Выборка из таблицы или представления . . . 202
Выборка из селективной хранимой процедуры . . . 203
Выборка из производной таблицы . . . 203
Выборка из общих табличных выражений . . . 204
11.1.6 JOIN . . . 205
Неявное соединение . . . 206
Внутреннее (INNER) соединение . . . 206
Внешние (OUTER) соединения . . . 207

Перекрестное (CROSS) соединение . . . 209
Квалифицированные соединения . . . 210
Естественное соединение . . . 212
Смешивание явного и неявного соединения . . . 212
Примечание о равенстве . . . 213
Неоднозначные имена полей в соединениях . . . 213
Соединения с хранимыми процедурами . . . 214
Соединение с производными таблицами . . . 214
11.1.7 WHERE . . . 215
11.1.8 GROUP BY . . . 216
HAVING . . . 219
11.1.9 WINDOW . . . 220
11.1.10 UNION . . . 221
11.1.11 PLAN . . . 223
Примеры простых планов . . . 224
Примеры составных планов . . . 227
11.1.12 ORDER BY . . . 233
Сортировка частей UNION . . . 234
11.1.13 OPTIMIZE FOR . . . 234
11.1.14 ROWS . . . 235
Соотношения между ключевыми словами FIRST и SKIP и предложением ROWS 236
Использование ROWS в UNION . . . 236
11.1.15 FETCH, OFFSET . . . 237
11.1.16 WITH LOCK . . . 237
Как сервер работает с WITH LOCK . . . 239
Предостережения при использовании WITH LOCK . . . 239
11.1.17 INTO . . . 240
11.1.18 Полный синтаксис SELECT . . . 240
11.2 INSERT . . . 243
11.2.1 VALUES . . . 245
11.2.2 Поиск многих . . . 245
11.2.3 DEFAULT VALUES . . . 246
11.2.4 OVERRIDING . . . 247
11.2.5 RETURNING . . . 247
11.3 UPDATE . . . 248
11.3.1 SET . . . 248
11.3.2 WHERE . . . 249
11.3.3 PLAN . . . 249
11.3.4 ORDER BY и ROWS . . . 250
11.3.5 RETURNING . . . 251
11.4 UPDATE OR INSERT . . . 251
11.5 DELETE . . . 252
11.5.1 WHERE . . . 253
11.5.2 PLAN . . . 253
11.5.3 ORDER BY и ROWS . . . 254
11.5.4 RETURNING . . . 254
11.6 MERGE . . . 255
11.7 EXECUTE PROCEDURE . . . 257
11.8 EXECUTE BLOCK . . . 257
11.9 SET OPTIMIZE . . . 260

12 Генераторы (GENERATOR/SEQUENCE) 261
12.1 Создание генератора . . . 261
12.2 Изменение значения генератора . . . 262

12.3 Создание нового или изменение существующего генератора . . . 263
12.4 Удаление генератора . . . 263
12.5 Пересоздание генератора . . . 264
12.6 Примечание к генератору . . . 264

13 BLOB фильтр (FILTER) 265
13.1 Объявление BLOB фильтра . . . 265
13.1.1 Задание подтипов . . . 265
13.1.2 Другие параметры . . . 266
13.2 Удаление объявления BLOB фильтра . . . 266

14 Индексы (INDEX) 267
14.1 Создание индекса . . . 267
14.1.1 Ограничения на индексы . . . 269
14.2 Изменение индекса . . . 269
14.3 Удаление индекса . . . 271
14.4 Селективность индекса . . . 272
14.5 Примечание индекса . . . 272

15 Представления (VIEW) 273
15.1 Создание представлений . . . 273
15.1.1 Поля представления . . . 273
15.1.2 Предложение AS . . . 274
15.1.3 Изменяемые представления . . . 274
15.1.4 Предложение WITH CHECK OPTION . . . 275
15.2 Изменение представлений . . . 275
15.3 Создание или изменение представлений . . . 276
15.4 Удаление представлений . . . 276
15.5 Пересоздание представлений . . . 276
15.6 Примеры представлений . . . 277
15.7 Преобразование неизменяемых представлений в изменяемые при помощи триггеров . . . 280
15.8 Системные представления . . . 283
15.9 Примечание представления . . . 286

16 Исключения (EXCEPTION) 287
16.1 Создание исключений . . . 287
16.2 Изменение исключений . . . 287
16.3 Создание или изменение исключений . . . 288
16.4 Пересоздание исключений . . . 288
16.5 Удаление исключений . . . 288

17 Процедурный язык PSQL 289
17.1 Элементы PSQL . . . 289
17.1.1 DML операторы с параметрами . . . 289
17.1.2 Транзакции . . . 289
17.1.3 Структура модуля . . . 289
Заголовок модуля . . . 290
Привилегии выполнения PSQL кода . . . 290
Тело модуля . . . 290
17.2 PSQL блоки . . . 292
17.3 Написание кода тела модуля . . . 292
17.3.1 Оператор присваивания . . . 293
17.3.2 Объявление локальной переменной . . . 294
Типы данных для переменных . . . 294
Ограничение NOT NULL . . . 295

Предложения CHARACTER SET и COLLATE . . . 295
Инициализация переменной . . . 295
Пример объявления локальной переменной . . . 295
17.3.3 Объявление курсора . . . 295
Однонаправленные и прокручиваемые курсоры . . . 296
Особенности использования курсора . . . 296
Пример использования именованного курсора . . . 297
17.4 Объявление и реализация подпроцедуры . . . 298
17.4.1 Объявление и реализация подфункции . . . 300
17.4.2 Определение составного оператора . . . 301
17.4.3 Оператор IF-THEN-ELSE . . . 301
17.4.4 Оператор WHILE-DO . . . 302
17.4.5 BREAK . . . 303
17.4.6 LEAVE . . . 303
17.4.7 CONTINUE . . . 305
17.4.8 EXIT . . . 305
17.4.9 SUSPEND . . . 306
17.4.10 EXECUTE STATEMENT . . . 306
Параметризованные операторы . . . 307
WITH {AUTONOMOUS | COMMON} TRANSACTION . . . 308
WITH CALLER PRIVILEGES . . . 308
ON EXTERNAL [DATA SOURCE] . . . 308
AS USER, PASSWORD и ROLE . . . 308
17.4.11 Пул внешних подключений . . . 309
17.4.12 FOR SELECT-DO . . . 311
Необъявленный курсор . . . 311
Пример . . . 312
17.4.13 FOR EXECUTE STATEMENT . . . 312
17.4.14 OPEN . . . 313
17.4.15 FETCH . . . 313
17.4.16 CLOSE . . . 314
17.4.17 IN AUTONOMOUS TRANSACTION . . . 314
17.4.18 Отправка событий базы данных . . . 315
17.4.19 RETURN . . . 315
17.5 Обработка ошибок . . . 316
17.5.1 Пользовательские исключения . . . 316
17.5.2 WHEN-DO . . . 317

18 Хранимые процедуры (PROCEDURE) 319
18.1 Создание хранимой процедуры . . . 319
18.1.1 Привилегии выполнения . . . 321
18.1.2 Параметры . . . 322
Входные параметры . . . 322
Выходные параметры . . . 322
Использование доменов при объявлении параметров . . . 322
Использование доменов при объявлении параметров . . . 322
Использование типа столбца при объявлении параметров . . . 323
18.1.3 Тело хранимой процедуры . . . 323
18.1.4 Внешние хранимые процедуры . . . 323
18.1.5 Привилегии выполнения . . . 323
18.1.6 Использование BLR вместо SQL кода . . . 323
18.2 Изменение хранимой процедуры . . . 324
18.3 Создание новой или изменение существующей хранимой процедуры . . . 325
18.4 Удаление хранимой процедуры . . . 326

18.5 Создание новой или пересоздание существующей хранимой процедуры . . . 326
18.6 Примеры хранимых процедур . . . 326
18.6.1 Получение значения искусственного первичного ключа . . . 327
18.6.2 Вычисление факториала числа . . . 327
18.7 Объявление подпроцедуры . . . 333

19 Хранимые функции (FUNCTION) 334
19.1 Создание хранимой функции . . . 334
19.1.1 Входные параметры . . . 335
19.1.2 Использование типа столбца при объявлении параметров . . . 336
19.1.3 Возвращаемое значение . . . 336
19.1.4 Детерминированные функции . . . 336
19.1.5 Тело хранимой функции . . . 336
Объявление локальных переменных, курсоров и подпрограмм . . . 336
19.1.6 Внешние функции . . . 337
19.1.7 Привилегии выполнения . . . 337
19.1.8 Использование BLR вместо SQL кода . . . 337
19.2 Изменение хранимой функции . . . 337
19.3 Создание новой или изменение существующей хранимой функции . . . 339
19.4 Удаление хранимой функции . . . 339
19.5 Создание новой или пересоздание существующей хранимой функции . . . 340
19.6 Объявление подфункции . . . 340

20 Триггеры (TRIGGER) 341
20.1 Создание триггера . . . 344
20.1.1 Состояние триггера . . . 346
20.1.2 Внешние триггеры . . . 346
20.1.3 Привилегии выполнения . . . 346
20.1.4 DML триггеры . . . 346
Фаза . . . 346
Порядок срабатывания . . . 347
События строк . . . 347
20.1.5 Триггеры базы данных . . . 347
20.2 Изменение триггера . . . 347
20.3 Создание нового или изменение существующего триггера . . . 348
20.4 Удаление триггера . . . 348
20.5 Создание нового или пересоздание существующего триггера . . . 349
20.6 Примеры триггеров . . . 349
20.6.1 Формирование значения искусственного первичного ключа . . . 349
20.6.2 Передача сообщений клиентским процессам об изменении данных . . . 350
20.6.3 Пример триггера, обеспечивающего поддержание ссылочной целостности данных 351
20.6.4 Пример триггера, создающего запись истории окладов . . . 351
20.6.5 Триггеры, преобразующие неизменяемые представления в изменяемые . . . 352

21 Пакеты (PACKAGE) 353
21.1 Создание заголовка пакета . . . 353
21.1.1 Привилегии выполнения . . . 354
21.1.2 Входные параметры . . . 355
21.1.3 Выходные параметры . . . 355
21.1.4 Использование доменов при объявлении параметров . . . 355
21.1.5 Использование типа столбца при объявлении параметров . . . 355
21.1.6 Детерминированные функции . . . 355
21.2 Изменение заголовка пакета . . . 356
21.3 Создание нового или изменение существующего заголовка пакета . . . 356

21.4 Удаление заголовка пакета . . . 357
21.5 Создание нового или пересоздание существующего заголовка объекта . . . 357
21.6 Создание тела пакета . . . 358
21.7 Удаление тела пакета . . . 359
21.8 Создание нового или пересоздание существующего тела объекта . . . 360

22 Сортировка (COLLATION) 361
22.1 Создание сортировки . . . 361
22.2 Удаление сортировки . . . 363

23 Планировщик заданий 364
23.1 Создание задания . . . 365
23.2 Изменение задания . . . 366
23.3 Удаление задания . . . 366
23.4 Оповещения . . . 366
23.5 Примечание для задания . . . 367

24 Внешние хранимые процедуры, функции и триггеры, написанные на языке Java 368
24.1 Объявление/изменение/пересоздание внешних процедур . . . 368
24.2 Объявление/изменение/пересоздание внешних функций . . . 369
24.3 Объявление/изменение/пересоздание внешних триггеров . . . 370
24.4 Удаление внешних процедур, функций и триггеров . . . 372
24.5 Вызов внешних процедур и функций . . . 372

25 Полнотекстовый поиск 373
25.1 Миграция полнотекстового поиска с версии 3.0 на 5.0 . . . 373
25.2 Настройка сервера для работы полнотекстового поиска . . . 374
25.2.1 Безопасность . . . 376
25.3 Служебные объекты для полнотекстового поиска . . . 377
25.3.1 Служебные домены . . . 377
25.3.2 Служебные таблицы . . . 377
FTS$INDICES . . . 377
FTS$INDEX_SEGMENTS . . . 378
FTS$POOL . . . 378
25.3.3 Служебные хранимые процедуры . . . 379
FTS$CREATE_INDEX . . . 379
FTS$DROP_INDEX . . . 380
FTS$ADD_FIELD_TO_INDEX . . . 380
FTS$DROP_FIELD_FROM_INDEX . . . 380
FTS$REINDEX . . . 381
FTS$SEARCH . . . 381
FTS$SEARCH_ID . . . 382
FTS$SEARCH_JSON . . . 382
FTS$FULL_REINDEX . . . 383
FTS$STARTDAEMON . . . 383
FTS$STOPDAEMON . . . 383
25.4 Пример использования полнотекстового поиска . . . 383
25.4.1 Создание индекса . . . 383
25.4.2 Добавление полей в индекс . . . 384
25.4.3 Удаление полей из индекса . . . 384
25.4.4 Переиндексация . . . 384
25.4.5 Поиск . . . 385
25.4.6 Индексация изменений в таблице . . . 386
25.4.7 Удаление индекса . . . 387
25.5 Синтаксис поисковых запросов . . . 387

25.5.1 Термы . . . 387
25.5.2 Маска . . . 388
25.5.3 Нечеткий поиск . . . 388
25.5.4 Усиление термов . . . 388
25.5.5 Логические операции . . . 388
Оператор OR . . . 388
Оператор AND . . . 389
Оператор + . . . 389
Оператор NOT . . . 389
Оператор — . . . 389
Группировка булевых операторов . . . 389
25.5.6 Экранирование специальных символов . . . 390

26 Операторы обеспечения безопасности 391
26.1 Операторы управления пользователями . . . 391
26.1.1 CREATE USER . . . 391
26.1.2 ALTER USER . . . 392
26.1.3 CREATE OR ALTER USER . . . 393
26.1.4 DROP USER . . . 394
26.1.5 RECREATE USER . . . 394
26.2 Операторы управления ролями . . . 395
26.2.1 CREATE ROLE . . . 395
26.2.2 ALTER ROLE . . . 397
26.2.3 ALTER ROLE RDB$ADMIN . . . 397
26.2.4 DROP ROLE . . . 397
26.3 Операторы отображения объектов безопасности . . . 398
26.3.1 CREATE MAPPING . . . 398
26.3.2 ALTER MAPPING . . . 400
26.3.3 CREATE OR ALTER MAPPING . . . 400
26.3.4 DROP MAPPING . . . 401
26.4 Операторы управления политиками . . . 401
26.4.1 CREATE POLICY . . . 401
26.4.2 ALTER POLICY . . . 402
26.4.3 DROP POLICY . . . 402
26.5 Операторы управления привилегиями . . . 402
26.5.1 GRANT . . . 403
Предложение TO . . . 404
WITH GRANT OPTION . . . 405
Предложение GRANTED BY . . . 405
Привилегии для таблиц и представлений . . . 405
Привилегии EXECUTE . . . 405
Привилегии USAGE . . . 405
Привилегии на выполнение DDL операций . . . 406
DDL привилегии на базу данных . . . 406
Назначение ролей . . . 406
Назначение политик . . . 407
Выдача прав системным привилегиям . . . 407
26.5.2 REVOKE . . . 407
Предложение FROM . . . 409
GRANT OPTION FOR . . . 409
ADMIN OPTION FOR . . . 409
GRANTED BY . . . 409
ALL ON ALL . . . 410

27 Сессионное окружение 411
27.1 Тайм-ауты . . . 411
27.1.1 Тайм-аут выполнения SQL оператора . . . 411
27.1.2 Тайм-аут простоя соединения . . . 412
27.2 Пул внешних соединений . . . 413
27.3 Изменение текущей роли . . . 414
27.4 Управление часовым поясом и обработкой часовых поясов . . . 414
27.5 Управление обработкой DECFLOAT . . . 414
27.6 Сброс сессионного окружения . . . 414
Приложение А Зарезервированные и ключевые слова . . . 416
Приложение Б Коды ошибок РЕД База Данных . . . 423
Б.1 Коды ошибок GDSCODE и SQLCODE . . . 423
Б.2 Коды ошибок SQLSTATE . . . 507
Приложение В Описание системных таблиц . . . 515
RDB$AUTH_MAPPING . . . 516
RDB$BACKUP_HISTORY . . . 517
RDB$CHARACTER_SETS . . . 517
RDB$CHECK_CONSTRAINTS . . . 518
RDB$COLLATIONS . . . 518
RDB$CONFIG . . . 519
RDB$CONSTANTS . . . 519
RDB$DATABASE . . . 520
RDB$DB_CREATORS . . . 520
RDB$DEPENDENCIES . . . 521
RDB$EXCEPTIONS . . . 522
RDB$FIELDS . . . 523
RDB$FIELD_DIMENSIONS . . . 526
RDB$FILES . . . 526
RDB$FILTERS . . . 526
RDB$FORMATS . . . 527
RDB$FUNCTIONS . . . 527
RDB$FUNCTION_ARGUMENTS . . . 528
RDB$GENERATORS . . . 530
RDB$INDEX_SEGMENTS . . . 531
RDB$INDICES . . . 531
RDB$JOBS . . . 532
RDB$JOBS_LOG . . . 533
RDB$KEYWORDS . . . 533
RDB$LOG_FILES . . . 534
RDB$PACKAGES . . . 534
RDB$PAGES . . . 535
RDB$PROCEDURE_PARAMETERS . . . 535
RDB$PROCEDURES . . . 536
RDB$REF_CONSTRAINTS . . . 537
RDB$RELATION_CONSTRAINTS . . . 537
RDB$RELATION_FIELDS . . . 538
RDB$RELATIONS . . . 539
RDB$ROLES . . . 540
RDB$SECURITY_CLASSES . . . 541
RDB$TABLESPACES . . . 541
RDB$TIME_ZONES . . . 542
RDB$TRANSACTIONS . . . 542
RDB$TRIGGER_MESSAGES . . . 542
RDB$TRIGGERS . . . 542

RDB$TYPES . . . 545
RDB$PUBLICATIONS . . . 545
RDB$PUBLICATION_TABLES . . . 546
RDB$USER_PRIVILEGES . . . 546
RDB$VIEW_RELATIONS . . . 549
Поле RDB$VALID_BLR . . . 549
Приложение Г Системные пакеты . . . 553
Г.1 RDB$TIME_ZONE_UTIL . . . 553
Г.1.1 Функция DATABASE_VERSION . . . 553
Г.1.2 Процедура TRANSITIONS . . . 553
Г.2 RDB$BLOB_UTIL . . . 554
Г.2.1 Функция NEW_BLOB . . . 554
Г.2.2 Функция OPEN_BLOB . . . 555
Г.2.3 Функция IS_WRITABLE . . . 555
Г.2.4 Функция READ_DATA . . . 555
Г.2.5 Функция SEEK . . . 556
Г.2.6 Процедура CANCEL_BLOB . . . 557
Г.2.7 Процедура CLOSE_HANDLE . . . 558
Приложение Д Наборы символов и порядки сортировки . . . 559
Приложение Е Функции, определенные пользователем (UDF) . . . 563
Е.1 Объявление в базе данных UDF . . . 563
Е.2 Изменение точки входа или имени модуля для UDF . . . 564
Е.3 Удаление объявления UDF из базы данных . . . 564
Приложение Ж Операторы SQL . . . 565
ALTER CHARACTER SET . . . 565
ALTER DATABASE . . . 565
ALTER DOMAIN . . . 566
ALTER EXCEPTION . . . 567
ALTER EXTERNAL CONNECTIONS POOL CLEAR ALL . . . 567
ALTER EXTERNAL CONNECTIONS POOL CLEAR OLDEST . . . 567
ALTER EXTERNAL CONNECTIONS POOL SET LIFETIME . . . 567
ALTER EXTERNAL CONNECTIONS POOL SET SIZE . . . 568
ALTER EXTERNAL FUNCTION . . . 568
ALTER FUNCTION . . . 568
ALTER INDEX . . . 570
ALTER JOB . . . 570
ALTER MAPPING . . . 571
ALTER PACKAGE . . . 571
ALTER POLICY . . . 572
ALTER PROCEDURE . . . 572
ALTER ROLE . . . 574
ALTER SEQUENCE . . . 574
ALTER SESSION RESET . . . 574
ALTER TABLE . . . 575
ALTER TABLESPACE . . . 577
ALTER TRIGGER . . . 578
ALTER USER . . . 578
ALTER VIEW . . . 579
CALL . . . 580
COMMENT . . . 581
COMMIT . . . 582
CONNECT . . . 582
CREATE COLLATION . . . 583
CREATE DATABASE . . . 583

CREATE DOMAIN . . . 584
CREATE EXCEPTION . . . 585
CREATE FUNCTION . . . 585
CREATE GENERATOR . . . 586
CREATE GLOBAL TEMPORARY TABLE . . . 587
CREATE INDEX . . . 587
CREATE JOB . . . 588
CREATE MAPPING . . . 588
CREATE OR ALTER EXCEPTION . . . 589
CREATE OR ALTER FUNCTION . . . 589
CREATE OR ALTER MAPPING . . . 590
CREATE OR ALTER PACKAGE . . . 590
CREATE OR ALTER PROCEDURE . . . 591
CREATE OR ALTER SEQUENCE . . . 592
CREATE OR ALTER TRIGGER . . . 592
CREATE OR ALTER USER . . . 594
CREATE OR ALTER VIEW . . . 595
CREATE PACKAGE . . . 595
CREATE PACKAGE BODY . . . 596
CREATE POLICY . . . 597
CREATE PROCEDURE . . . 598
CREATE ROLE . . . 599
CREATE SEQUENCE . . . 600
CREATE SHADOW . . . 600
CREATE TABLE . . . 601
CREATE TABLESPACE . . . 602
CREATE TRIGGER . . . 603
CREATE USER . . . 605
CREATE VIEW . . . 606
DECLARE EXTERNAL FUNCTION . . . 606
DECLARE FILTER . . . 607
DELETE . . . 608
DROP COLLATION . . . 608
DROP DATABASE . . . 609
DROP DOMAIN . . . 609
DROP EXCEPTION . . . 609
DROP EXTERNAL FUNCTION . . . 610
DROP FILTER . . . 610
DROP FUNCTION . . . 610
DROP GENERATOR . . . 610
DROP INDEX . . . 611
DROP JOB . . . 611
DROP MAPPING . . . 611
DROP PACKAGE . . . 612
DROP PACKAGE BODY . . . 612
DROP POLICY . . . 612
DROP PROCEDURE . . . 612
DROP ROLE . . . 613
DROP SEQUENCE . . . 613
DROP SHADOW . . . 613
DROP TABLE . . . 614
DROP TABLESPACE . . . 614
DROP TRIGGER . . . 614
DROP USER . . . 615

DROP VIEW . . . 615
EXECUTE BLOCK . . . 616
EXECUTE PROCEDURE . . . 616
GRANT . . . 617
INSERT . . . 619
MERGE . . . 619
RECREATE EXCEPTION . . . 620
RECREATE FUNCTION . . . 621
RECREATE PACKAGE . . . 621
RECREATE PACKAGE BODY . . . 622
RECREATE PROCEDURE . . . 623
RECREATE SEQUENCE . . . 624
RECREATE TABLE . . . 624
RECREATE TRIGGER . . . 625
RECREATE USER . . . 626
RECREATE VIEW . . . 627
RELEASE SAVEPOINT . . . 627
REVOKE . . . 628
ROLLBACK . . . 629
SAVEPOINT . . . 630
SELECT . . . 630
Предложение OPTIMIZE FOR . . . 631
SET BIND . . . 632
SET DECFLOAT BIND . . . 633
SET DECFLOAT ROUND . . . 634
SET DECFLOAT TRAPS TO . . . 635
SET GENERATOR . . . 636
SET NAMES . . . 636
SET ROLE . . . 637
SET SESSION IDLE TIMEOUT . . . 637
SET SQL DIALECT . . . 637
SET STATEMENT TIMEOUT . . . 637
SET STATISTICS . . . 638
SET TIME ZONE . . . 638
SET TIME ZONE BIND . . . 639
SET TRANSACTION . . . 639
SET TRUSTED ROLE . . . 640
SIMILAR TO . . . 640
SUSPEND . . . 641
UPDATE . . . 642
UPDATE OR INSERT . . . 643
Приложение З Встроенные функции . . . 644
З.1 Скалярные функции . . . 644
З.1.1 Функции для работы с контекстными переменными . . . 644
RDB$GET_CONTEXT . . . 644
RDB$SET_CONTEXT . . . 647
З.1.2 Функции для работы с файлами . . . 648
CREATE_FILE . . . 648
DELETE_FILE . . . 649
READ_FILE . . . 649
З.1.3 Функции подсистемы безопасности . . . 649
CHECK_DDL_RIGHTS . . . 649
CHECK_DML_RIGHTS . . . 650
RDB$ROLE_IN_USE . . . 650

RDB$SYSTEM_PRIVILEGE . . . 650
MAKE_DBKEY . . . 651
3.1.4 RDB$TRACE_MSG . . . 652
3.1.5 Функции для обработки ошибок . . . 652
RDB$ERROR . . . 652
3.1.6 Функции работы с транзакциями . . . 653
RDB$GET_TRANSACTION_CN . . . 653
3.1.7 Математические функции . . . 653
ABS . . . 653
ACOS . . . 653
ACOSH . . . 654
ASIN . . . 654
ASINH . . . 654
ATAN . . . 654
ATAN2 . . . 655
ATANH . . . 655
CEIL|CEILING . . . 655
COS . . . 656
COSH . . . 656
COT . . . 656
EXP . . . 656
FLOOR . . . 656
LN . . . 657
LOG . . . 657
LOG10 . . . 657
MOD . . . 657
PI . . . 658
POWER . . . 658
RAND . . . 658
ROUND . . . 658
SIGN . . . 658
SIN . . . 659
SINH . . . 659
SQRT . . . 659
TAN . . . 659
TANH . . . 659
TRUNC . . . 660
3.1.8 Функции для работы со строками . . . 660
ASCII_CHAR . . . 660
ASCII_VAL . . . 660
BIT_LENGTH . . . 661
CHARACTER_LENGTH . . . 661
DAMLEV . . . 662
LEFT . . . 662
LOWER . . . 662
LPAD . . . 663
OCTET_LENGTH . . . 663
OVERLAY . . . 664
POSITION . . . 665
REGEXP_SUBSTR . . . 665
REPLACE . . . 667
REVERSE . . . 667
RIGHT . . . 667
RPAD . . . 667

SUBSTRING . . . 668
Позиционный SUBSTRING . . . 669
SUBSTRING по регулярному выражению . . . 670
TRIM . . . 670
UPPER . . . 671
UNICODE_CHAR . . . 672
UNICODE_VAL . . . 672
3.1.9 Функции для работы с датой и временем . . . 672
AT . . . 672
DATEADD . . . 672
DATEDIFF . . . 673
EXTRACT . . . 674
FIRST_DAY . . . 674
LAST_DAY . . . 675
UTC_TIMESTAMP . . . 675
3.1.10 Функции для работы с типом DECFLOAT . . . 675
COMPARE_DECFLOAT . . . 675
NORMALIZE_DECFLOAT . . . 676
QUANTIZE . . . 676
TOTALORDER . . . 677
3.1.11 Кодирование и декодирование бинарных данных . . . 677
BASE64_ENCODE . . . 677
BASE64_DECODE . . . 678
HEX_ENCODE . . . 678
HEX_DECODE . . . 678
BLOB_APPEND . . . 678
3.1.12 Хэш функции . . . 680
HASH . . . 680
CRYPT_HASH . . . 680
HASH_CP . . . 681
HASHAGG . . . 681
3.1.13 Криптографические функции . . . 681
CRC32 . . . 681
ENCRYPT . . . 682
DECRYPT . . . 683
RSA_PRIVATE . . . 684
RSA_PUBLIC . . . 684
RSA_ENCRYPT . . . 684
RSA_DECRYPT . . . 685
RSA_SIGN . . . 685
RSA_VERIFY . . . 686
3.1.14 Функции преобразования типов . . . 686
CAST . . . 686
3.1.15 Функции побитовых операций . . . 687
BIN_AND . . . 687
BIN_NOT . . . 687
BIN_OR . . . 687
BIN_SHL . . . 688
BIN_SHR . . . 688
BIN_XOR . . . 689
3.1.16 Функции для работы с UUID . . . 689
CHAR_TO_UUID . . . 689
GEN_UUID . . . 689
UUID_TO_CHAR . . . 690

3.1.17 Функции для работы с генераторами . . . 690
GEN_ID . . . 690
NEXT VALUE FOR . . . 691
3.1.18 Условные функции . . . 691
CASE-WHEN-ELSE . . . 691
COALESCE . . . 692
DECODE . . . 692
IIF . . . 693
MAXVALUE . . . 693
MINVALUE . . . 693
NULLIF . . . 693
GREATEST . . . 693
LEAST . . . 694
3.1.19 Функции мониторинга ЦП . . . 694
CPU_LOAD . . . 694
3.1.20 Функции для работы с LDAP . . . 694
LDAP_ATTR . . . 694
3.2 Агрегатные функции . . . 694
3.2.1 Предложение FILTER . . . 695
3.2.2 AVG . . . 696
3.2.3 COUNT . . . 696
3.2.4 LIST . . . 697
3.2.5 MAX . . . 698
3.2.6 MIN . . . 698
3.2.7 SUM . . . 699
3.3 Статистические функции . . . 699
3.3.1 CORR . . . 699
3.3.2 COVAR_POP . . . 700
3.3.3 COVAR_SAMP . . . 700
3.3.4 STDDEV_POP . . . 701
3.3.5 STDDEV_SAMP . . . 701
3.3.6 VAR_POP . . . 702
3.3.7 VAR_SAMP . . . 702
3.4 Функции линейной регрессии . . . 703
3.4.1 REGR_AVGX . . . 703
3.4.2 REGR_AVGY . . . 703
3.4.3 REGR_COUNT . . . 704
3.4.4 REGR_INTERCEPT . . . 704
3.4.5 REGR_R2 . . . 705
3.4.6 REGR_SLOPE . . . 705
3.4.7 REGR_SXX . . . 705
3.4.8 REGR_SXY . . . 706
3.4.9 REGR_SYY . . . 706
3.5 Оконные функции . . . 707
3.5.1 Агрегатные функции . . . 708
3.5.2 Секционирование . . . 709
3.5.3 Сортировка . . . 710
3.5.4 Рамка окна . . . 712
Окна диапазона . . . 713
Окна строк . . . 714
3.5.5 Именованные окна . . . 714
3.5.6 Ранжирующие функции . . . 715
DENSE_RANK . . . 715
RANK . . . 716

PERCENT_RANK . . . 716
CUME_DIST . . . 717
NTILE . . . 718
ROW_NUMBER . . . 718
3.5.7 Навигационные функции . . . 719
FIRST_VALUE . . . 720
LAG . . . 720
LAST_VALUE . . . 721
LEAD . . . 721
NTH_VALUE . . . 721
3.5.8 Агрегатные функции внутри оконных . . . 722
3.6 Табличные функции . . . 722
3.6.1 UNLIST . . . 722
Приложение И Контекстные переменные . . . 724
CURRENT_CONNECTION . . . 724
CURRENT_DATE . . . 724
CURRENT_ROLE . . . 724
CURRENT_TIME . . . 724
CURRENT_TIMESTAMP . . . 724
CURRENT_TRANSACTION . . . 725
CURRENT_USER . . . 725
DECFLOAT_ROUND . . . 725
DECFLOAT_TRAPS . . . 725
INSERTING, UPDATING и DELETING . . . 725
LOCALTIME . . . 725
LOCALTIMESTAMP . . . 726
NEW . . . 726
NOW . . . 726
OLD . . . 727
ROW_COUNT . . . 727
REPLICATION_STATUS_JSON . . . 727
SQLCODE, GDSCODE . . . 728
SQLSTATE . . . 728
TODAY . . . 729
TOMORROW . . . 729
USER . . . 729
YESTERDAY . . . 729
Приложение К Операторы языка хранимых процедур, функций и триггеров . . . 730
BREAK . . . 730
CLOSE . . . 730
CONTINUE . . . 730
DECLARE CURSOR . . . 731
DECLARE FUNCTION . . . 731
DECLARE PROCEDURE . . . 732
DECLARE VARIABLE . . . 733
EXCEPTION . . . 733
EXIT . . . 734
FETCH . . . 734
FOR EXECUTE STATEMENT . . . 736
FOR SELECT-DO . . . 737
IF-THEN-ELSE . . . 738
LEAVE . . . 738
OPEN . . . 738
POST_EVENT . . . 738

SUSPEND 739
WHEN-DO 739
WHILE-DO 740
Приложение Л Список региональных часовых поясов 741

Введение

Настоящий документ «Руководство по SQL» содержит полное описание языковых средств, используемых для работы с системой управления базами данных РЕД База Данных. Он предназначен для администраторов баз данных, для разработчиков, проектирующих базы данных, и для программистов, пишущих программы, работающие с данными из базы данных. Использование настоящего документа не требует предварительного знакомства с другими документами.

Документ содержит множество примеров, иллюстрирующих применение языковых конструкций SQL.

В документе 28 глав и 11 приложений.

Глава 1 содержит общие сведения о реляционных базах данных, в нем приводится список объектов базы данных РЕД База Данных, описывается структура языка и основные синтаксические конструкции SQL, дается краткая информация о структуре данных на диске (On-Disk Structure, ODS) и о диалектах базы данных.

В главе 2 рассматриваются элементы, которые являются общими для всех реализаций языка SQL — выражения, которые используются для извлечения и работают на утверждениях о данных, и предикатов, которые проверяют истинность этих утверждений.

Глава 3 содержит подробные сведения об используемых в SQL типах данных, описываются допустимые операции над различными типами данных, средства преобразования одних типов данных в другие, приводится список предварительно определенных литералов и контекстных переменных.

Глава 5 содержит детальное описание характеристик транзакций — режимов доступа, уровней изоляции, режимов разрешения блокировок, средств резервирования. Приводится синтаксис операторов работы с транзакциями.

В главе 6 описываются языковые средства по созданию, изменению и удалению баз данных, подключение к существующей базе данных, удаление базы данных, приводится множество примеров создания однофайловых и многофайловых баз данных.

В главе 7 описываются языковые средства создания и удаления оперативных копий (shadow).

В главе 8 подробно рассматриваются языковые средства создания, изменения и удаления доменов.

В главе 9 описываются языковые средства для работы с таблицами базы данных и с генераторами. Описывается синтаксис и семантика операторов создания, изменения и удаления таблиц. Подробно описываются ограничения столбца и таблицы. Рассматриваются встроенные функции SQL. Вкратце рассматриваются вопросы выбора первичных ключей таблиц.

Глава 11 посвящена операторам, позволяющим вносить изменения в существующие таблицы базы данных. Рассматривается множество примеров добавления, изменения и удаления данных. Подробно рассматривается наиболее сложный и мощный оператор SQL, осуществляющий выборку данных. Детально рассматриваются все предложения оператора, в том числе, предложение `PLAN`, позволяющее задать пользовательский план выборки данных. Приводится большое количество примеров сложной выборки и упорядочения данных, выполнения объединения (`UNION`), соединения (`JOIN`) данных из нескольких таблиц, различных способов ограничения количества выводимых данных.

В главе 12 описываются средства работы с генераторами — создание, удаление, изменение текущего значения генераторов.

В главе 14 описывается порядок использования индексов для таблиц. Рассматриваются операторы создания индекса, изменения активности, удаления и улучшения селективности (избирательности) индекса.

В главе 15 рассматривается работа с представлениями (view). Приводится описание операторов создания и удаления представлений. Подробно описываются средства, позволяющие перевести неизменяемые представления в изменяемые. Даются примеры создания и использования различных представлений. Приводятся тексты нескольких системных представлений.

В главе 16 описываются операторы создания, модификации и удаления исключений, которые можно вызывать и обрабатывать в PSQL коде.

В главе 17 описывается язык хранимых процедур, функций, пакетов и триггеров PSQL. Здесь описывается объявления локальных переменных и курсоров, операторы присваивания, условные операторы, операторы циклов, выброса пользовательского исключений, средства для обработки ошибок, отправки сообщений (событий) клиентским программам и многое другое.

Глава 18 содержит описание операторов создания, модификации и удаления хранимых процедур. Приводятся примеры различных выполняемых хранимых процедур, хранимых процедур выбора.

Глава 19 содержит описание операторов создания, модификации и удаления хранимых функций.

Глава 20 содержит описание операторов создания, модификации и удаления триггеров. Различают табличные триггеры, DDL триггеры и триггеры на события базы данных. Приводятся примеры различных триггеров.

В главе 21 описываются операторы создания, модификации и удаления заголовков и тел пакетов.

В главе 23 дается описание встроенного в СУБД планировщика заданий, который создан, чтобы отменить необходимость использования средств ОС для создания заданий, выполняемых по расписанию. Описываются операторы создания, модификации и удаления планировщика.

В главе 24 описывается синтаксис операторов создания, изменения и удаления внешних хранимых процедур, функций и триггеров, написанных на Java. Приводятся примеры.

В главе 25 дается полное описание средств полнотекстового поиска, которые реализованы в РЕД Базе Данных.

В главе Приложение А приводятся списки ключевых (используемых в лексике, в словарном запасе SQL) слов и списки зарезервированных слов (слов, которые не могут быть использованы для имен объектов базы данных или переменных и параметров в хранимых процедурах или триггерах).

Глава Приложение Б содержит список кодов ошибок, которые могут возникнуть при работе с базой данных.

Глава Приложение В содержит список всех системных таблиц, в которых хранятся метаданные объектов базы данных.

В главе Приложение Д приводится список доступных наборов символов. Для каждого набора символов перечисляются порядки сортировки.

В главе Приложение Е описываются функции, определенные пользователем, которые поставляются вместе с системой. Задается форма обращения к этим функциям.

Глава Приложение Ж содержит краткое описание синтаксиса и семантики всех операторов. Операторы располагаются в алфавитном порядке. Это приложение может служить кратким справочником по операторам SQL.

Глава Приложение З содержит краткое описание синтаксиса и семантики всех встроенных функций. Функции располагаются в алфавитном порядке. Это приложение может служить кратким справочником по функциям SQL.

Глава Приложение И содержит краткое описание синтаксиса и семантики всех контекстных переменных SQL. Переменные располагаются в алфавитном порядке. Это приложение может служить кратким справочником по контекстным переменным SQL.

Глава Приложение Г содержит список всех системных пакетов. На данный момент содержит описание пакета для работы с часовыми поясами.

Глава Приложение К содержит краткое описание синтаксиса и семантики всех языковых конструкций, допустимых только в языке хранимых процедур и триггеров. Операторы располагаются в алфавитном порядке. Это приложение может служить кратким справочником по PSQL.

Глава Приложение Л содержит список региональных часовых поясов и их идентификаторов.

Глава 1

SQL РЕД Базы Данных. Общие сведения

В реляционной базе данных хранятся собственно обрабатываемые клиентскими программами данные и метаданные — описания этих данных. Для работы как с данными, так и с метаданными используется язык SQL (Structured Query Language) - структурированный язык запросов.

В реляционных базах данных все данные хранятся в таблицах. Это один из основных объектов базы данных. Метаданные, описания объектов реляционной базы данных, также хранятся в таблицах, в системных таблицах.

Базы данных используются для решения задач в конкретных областях человеческой деятельности. Это могут быть, например, задачи любого весьма сложного информационного поиска, расчета заработной платы, выплат в бюджеты разных уровней и т.д.

1.1 Структура данных на внешнем носителе (ODS)

С точки зрения операционной системы база данных — это файл или группа файлов, хранящихся на внешних носителях. Файл базы данных в РЕД Базе Данных имеет страничную довольно сложную организацию.

В разных версиях РЕД Базы Данных используются различные форматы хранения данных на внешних носителях. Такие форматы называются ODS (On-Disk Structure — структура данных на диске). Форматам присваиваются номера. В настоящей версии РЕД Базы Данных используется ODS версии 13.0. Узнать версию ODS можно с помощью `gstat -h`.

Таблица 1.1 — Соответствие версий ODS версиям РЕД Базы Данных

Версия СУБД	Версия ODS
РЕД База Данных 5.0.0	13.1
РЕД База Данных 3.0.8 - РЕД База Данных 3.0.10	12.3
РЕД База Данных 3.0.0 - РЕД База Данных 3.0.7	12.0
РЕД База Данных 2.6	2.4
РЕД База Данных 2.5	24578.3
РЕД База Данных 2.1	24578.2
РЕД База Данных 2.0	11.0

Как правило, не существует полной обратной совместимости более поздней версии ODS с более ранней. Чтобы перевести существующую базу данных с более ранней ODS в текущую необходимо выполнить резервное копирование базы данных с использованием соответствующего средства предыдущей версии (обычно это утилита командной строки `gbak`), а затем восстановить резервную копию в текущей версии РЕД Базы Данных. Обновить базу данных до последней минорной версии ODS в пределах поддерживаемой мажорной версии можно командой `gfix -upgrade`.

1.2 Объекты базы данных

Объектами базы данных в СУБД РЕД База Данных являются следующие.

Таблица (`table`)
: Это основной объект любой реляционной базы данных, в том числе, и РЕД Базы Данных. В таблицах хранятся все данные и метаданные базы данных. Таблица — это плоская двумерная структура, содержащая произвольное количество строк (`row`). Часто строку называют записью (`record`). Таблица может не содержать и ни одной строки — может быть пустой. Все строки одной таблицы имеют одинаковую структуру. Они состоят из столбцов (`column`). Другое название для столбца — поле (`field`). Таблица в реляционной базе данных должна содержать хотя бы один столбец. Каждый столбец имеет конкретный тип данных (`datatype`). Подробнее о типах данных РЕД Базы Данных см. в главе 3. О создании и изменении таблиц см. в главе 9.

Домен (`domain`)
: Объект базы данных, описывающий некоторые характеристики столбца. На домен можно ссылаться при описании столбцов создаваемой таблицы или при изменении характеристик столбцов существующей в базе данных таблицы. В этом случае все характеристики домена копируются в столбец. Некоторые характеристики домена при копировании в столбец могут быть изменены в описании столбца. На домены также можно ссылаться при описании параметров хранимых процедур, внутренних переменных хранимых процедур и триггеров. Домены описаны в главе 8.

Индекс (`index`)
: Объект базы данных, предназначенный для ускорения выборки данных из таблицы и/или для ускорения упорядочения результатов выборки данных из таблицы. Каждый индекс создается для одной конкретной таблицы. Индекс представляет собой множество упорядоченных строк, каждая из которых содержит значение полей, входящих в состав индекса, и указатель на строку таблицы, содержащую соответствующие значения этих полей. Существуют средства для активации/деактивации индексов, улучшения их характеристик — селективности (избирательности). Для первичных, уникальных и внешних ключей система автоматически создает индексы. Подробности см. в главе 14.

Генератор (`generator` **или** `sequence`)
: Простой объект реляционной базы данных, предназначенный для получения уникального числового значения, используемого, как правило, для формирования значения искусственного первичного ключа или иногда уникального ключа. Использование генераторов подробно описано в главе 12.

Хранимая процедура (`stored procedure`)
: Программа, написанная на процедурном расширении языка SQL (который также называется языком хранимых процедур и триггеров, PSQL), и хранящаяся в области метаданных базы данных, позволяющая выполнять различные действия с данными в базе данных. К хранимым процедурам могут обращаться хранимые процедуры этой базы данных, пользовательские (клиентские) программы и триггеры (см. ниже). Для хранимых процедур допустима также рекурсия — обращение процедуры к самой себе. Хранимая процедура выполняется на стороне сервера, что во многих случаях может резко сократить сетевой трафик и заметно увеличить скорость решения задач предметной области.

  Хранимые процедуры могут получать от вызывающей программы (хранимой процедуры, клиентской программы, триггера) параметры и возвращать произвольное количество значений. Существуют хранимые процедуры выбора, осуществляющие выбор данных из таблиц базы данных (к таким процедурам можно обращаться как и к обычным таблицам в операторе `SELECT`), и выполняемые хранимые процедуры, осуществляющие любые действия с данными. Использование хранимых процедур см. в главе 18. В РЕД Базе Данных также существуют дополнительные расширения языка хранимых процедур с добавлением элементов языка Java. Подробности см. в главе 24.

Хранимая функция (`stored function`)
: Программа, написанная на процедурном расширении языка SQL (который также называется язы-

ком хранимых процедур и триггеров, PSQL), и хранящаяся в области метаданных базы данных и выполняющаяся на стороне сервера. К хранимой функции могут обращаться хранимые процедуры, хранимые функции (в том числе и сама к себе), триггеры и клиентские программы. При обращении хранимой функции самой к себе такая хранимая функция называется рекурсивной. В отличие от хранимых процедур хранимые функции всегда возвращают одно скалярное значение. Использование хранимых функций см. в главе 19.

Триггер (`trigger`)
: Как и хранимая процедура является программой, написанной на процедурном расширении языка PSQL, хранящейся в области метаданных и выполняемой на сервере. Однако обращение напрямую к триггеру невозможно. Он автоматически вызывается при наступлении одной из фаз события, связанного с изменением данных в таблицах, или события, связанного с подключением к базе данных и с работой с транзакциями. События таблицы — добавление данных, изменение строки таблицы, удаление строки. Фазами являются «до» (`before`) — до выполнения действия — и «после» (`after`) — после выполнения действия. В РЕД Базе Данных один и тот же триггер может вызываться при наступлении одной из фаз сразу нескольких событий. Возможны пять вариантов вызова триггера, которые связаны с процессом подключения к базе данных и с работой с транзакциями. Подробности об использовании триггеров см. в главе 20.

Пакет (`package`)
: Группа процедур и функций, которая представляет собой единый объект базы данных. Пакеты упрощают механизм отслеживания зависимостей между набором связанных процедур, а также между этим набором и другими процедурами, как упакованными, так и неупакованными. Подробности об использовании пакетов см. в главе 21.

Пользовательские исключения (`exception`)
: Объект реляционной базы данных, который позволяет создавать, а затем выдавать сообщения пользователю при появлении некоторой ситуации в процессе работы программ с базой данных. Это могут быть как ошибочные ситуации, возникающие при каких-либо нарушениях в базе данных, так и любые другие особые случаи обработки данных в базе данных. Эти исключения могут использоваться только в хранимых процедурах и триггерах. Пользовательские исключения описываются в главе 16 и в разделе 17.5.

События базы данных (`event`)
: В РЕД Базе Данных существует возможность из хранимых процедур и триггеров передавать некоторые сообщения всем клиентским приложениям, работающим с конкретной базой данных и «прослушивающим» данное сообщение. Это средство позволяет во многих случаях осуществлять синхронизацию отображения данных либо выполнять более сложные действия по взаимодействию клиентских приложений при их совместной одновременной работе с базой данных. События базы данных описываются в главе 17.

Представление (`view`)
: Другие названия — обзор, просмотр. Это результат выборки данных из одной или более таблиц базы данных на основании конкретных часто довольно сложных критериев. Основой представления является оператор `SELECT` любой сложности. Представление является как бы виртуальной таблицей, которая на самом деле не хранится в базе данных. Представление — удобное средство для получения данных в случае достаточно сложной выборки данных, когда от пользователя нужно скрывать сложные условия такой выборки. Кроме того, представления являются полезным средством скрыть от пользователя некоторые столбцы таблиц, значения которых не предназначены для просмотра этим пользователем. Представления описаны в главе 15.

Функции, определенные пользователем (`User Defined Functions, UDF`)

> Поддержка внешних UDF функций (реализованных в динамических библиотеках и определенных в базе данных с помощью функций оператора `DECLARE EXTERNAL FUNCTION`) в версии 5.0 устарела.

Функции, написанные на любом языке программирования, и хранящиеся вне базы данных, но описанные в этой базе. Могут использоваться для расширения возможностей языка SQL и соответствующих языков программирования. Часто позволяют описывать довольно сложные действия с данными из базы данных (или с данными, передаваемыми в виде входных параметров). В РЕД Базе Данных представлено достаточно большое количество полезных функций, созданных разработчиками системы. Подробности описания в базе данных и использования `UDF` см. Приложение E.

Транзакция (`transaction`)

Это не объект базы данных, а некоторый механизм, используемый при обращении к данным или метаданным базы данных. Любые действия с данными (и метаданными) в базе данных выполняются в контексте (под управлением) какой-либо транзакции. В процессе «жизни» транзакции действия с данными базы данных, выполненные под управлением этой транзакции до ее подтверждения, не видны другим параллельным процессам. Все действия, выполненные в контексте транзакции, можно либо подтвердить (тогда изменения становятся видимыми в других параллельных процессах) или отменить, выполнить откат транзакции (тогда база данных возвращается в то состояние, которое она имела до старта этой транзакции). Существуют средства использовать вложенные транзакции, когда создаются определенные контрольные точки, и откат транзакции можно выполнить не на самое начало транзакции, а на определенную контрольную точку.

В РЕД Базе Данных используется многоверсионная архитектура (Multigenerational architecture, MDA). При такой архитектуре в результате добавления, изменения или удаления отдельных записей создаются новые версии этих записей. Благодаря этой архитектуре другие пользователи могут видеть и использовать предыдущие версии записи, даже если она была изменена или удалена в текущей транзакции.

Транзакциям посвящена глава 5.

1.3 Подмножества языка

В SQL РЕД Базы Данных выделяются следующие подмножества, подразделы языка.

Язык описания данных — DDL, Data Definition Language. При помощи этого подмножества языка создаются, изменяются и удаляются объекты базы данных, метаданные. Для создания нового экземпляра любого объекта базы данных используется оператор `CREATE`, для изменения — `ALTER`, для удаления — `DROP`.

Язык манипулирования данными — DML, Data Manipulation Language. Средства этого подмножества используются для изменения и выборки данных, хранящихся в базе данных, а также для запуска, подтверждения или отката транзакций. Для добавления новых данных (новых строк) в таблицы используется оператор `INSERT`, для изменения существующих данных — `UPDATE`, для удаления — `DELETE`, для выборки данных — `SELECT`. Для старта, запуска, транзакции используется оператор `SET TRANSACTION`, для ее подтверждения `COMMIT`, для отката транзакции — `ROLLBACK`.

Основная особенность языка SQL — его декларативность. При помощи большинства языковых средств пользователь задает, что должно быть сделано, но не указывает, как это должно быть выполнено. В подмножестве DML можно выделить декларативное ядро — языковые средства, используемые в любых ситуациях.

Существует также императивное расширение языковых средств DML, называемое процедурным SQL (PSQL) или языком хранимых процедур и триггеров. Оба термина являются синонимами. Это расширение содержит операцию присваивания, условные операторы, операторы циклов, средства обработки ошибок базы данных, выдачи исключений, отправки сообщений (событий) другим клиентским программам, работающим в настоящий момент с этой базой данных, контекстные переменные `NEW, OLD`, и некоторые другие средства, используемые в обычных языках программирования. Язык хранимых процедур и триггеров подробно описывается в главе 17.

В отдельное подмножество можно также выделить и группу операторов, связанных с управлением безопасностью. Это операторы создания, изменения и удаления учетных записей пользователей, предоставления и отмены привилегий пользователей базы данных, создания и удаления ролей, ис-

пользования политик безопасности и т.д. Привилегии и все соответствующие операторы описываются в документе «Руководство администратора».

1.4 Диалекты базы данных

В СУБД РЕД База Данных существует понятие диалекта базы данных. Каждый клиент, соединенный с базой данных, и каждая база данных имеют свой диалект SQL. Диалект определяет допустимость некоторых синтаксических конструкций в SQL, интерпретацию отдельных синтаксических конструкций и способ хранения столбцов различных типов данных на внешних носителях.

В РЕД Базе Данных поддерживается и диалект 1, который присутствовал в базах данных InterBase версии 5.6 и более ранних.

Основным диалектом в настоящее время является диалект 3, который более всего соответствует стандартам SQL и более удобен в работе. Новую базу данных СУБД РЕД База Данных создаёт в 3-м диалекте.

Для задания диалекта клиента используется оператор `SET SQL DIALECT`. Его синтаксис:

```
SET SQL DIALECT {1 | 3};
```

Установленный для клиента диалект присваивается создаваемой этим клиентом новой базе данных.

Основные отличия диалектов 1 и 3 приведены в таблице 1.2:

Таблица 1.2 — Отличия диалектов 1 и 3

Средство	Диалект 1	Диалект 3
Имена с разделителями, заключенные в кавычки "	Рассматриваются как символьные константы	Рассматриваются как идентификаторы, которые являются зависимыми от регистра
Тип данных `DATE`	Занимает 8 байтов. Содержит дату и время	Занимает 4 байта. Содержит только дату
Тип данных `TIMESTAMP`	Не используется	Занимает 8 байтов. Содержит дату и время
Типы данных `NUMERIC` **и** `DECIMAL`**, размер 1 ÷ 4**	`NUMERIC` хранится как `SMALLINT`, `DECIMAL` — как `INTEGER`	Оба хранятся как `SMALLINT`
Типы данных `NUMERIC` **и** `DECIMAL`**, размер 5 ÷ 9**	Хранятся как `INTEGER`	Хранятся как `INTEGER`
Типы данных `NUMERIC` **и** `DECIMAL`**, размер 10 ÷ 18**	Хранятся как `DOUBLE PRECISION`	Хранятся как `BIGINT`
Тип данных `BIGINT`	Недоступен	Доступен в качестве целого 64-х битного типа данных
Генераторы	Значение генераторов хранится как 64 битное целое, а при выдаче значения усекается до 32 битного целого	Значения генераторов хранятся как 64-ти битные целые значения

Существует еще диалект 2. Он применим только на стороне клиента. Этот диалект может быть использован лишь для проверки возможности миграции данных из более раннего диалекта 1 в диалект 3. В случае возможных ошибок выдается предупреждающее сообщение.

В настоящем документе при описании синтаксических конструкций используется только SQL диалекта 3.

1.5 Операторы. Общие сведения

Основная конструкция SQL — оператор (`statement`). Оператор описывает, что должна выполнить система управления базами данных с конкретным объектом данных или метаданных, обычно не указывая, как именно это должно быть выполнено. Достаточно сложные операторы содержат более простые конструкции — предложения (`clause`) и варианты, альтернативы. Предложение описывает некую законченную конструкцию в операторе. Например, предложение `WHERE` в операторе `SELECT` и в ряде других операторов (`UPDATE, DELETE`) задает условия поиска данных в таблице (таблицах), подлежащих выборке, изменению, удалению. Предложение `ORDER BY` задает характеристики упорядочения выходного, результирующего, набора данных. Альтернативы, будучи наиболее простыми конструкциями, задаются при помощи конкретных ключевых слов и определяют некоторые дополнительные характеристики элементов предложения (допустимость дублирования данных, варианты использования и др.).

1.6 Ключевые слова. Общие сведения

В SQL существуют ключевые слова и зарезервированные слова. Ключевые слова — это все слова, входящие в лексику (словарь) языка SQL. Ключевые слова можно использовать и в качестве имен, идентификаторов, объектов базы данных, внутренних переменных и параметров. Зарезервированные слова — это те ключевые слова, которые нельзя использовать в качестве имен объектов базы данных, переменных или параметров.

Список зарезервированных и ключевых слов представлен в Приложение А.

1.7 Идентификаторы

Все объекты базы данных имеют имена, которые иногда называют идентификаторами. Максимальная длина идентификатора составляет 63 символа. Существует два типа имен — имена, похожие по форме на имена переменных в обычных языках программирования, и имена с разделителями (`delimited name`), которые являются отличительной особенностью языка SQL.

Обычное имя должно начинаться с буквы латинского алфавита, за которой могут следовать буквы (латинского алфавита), цифры, символ подчеркивания и знак доллара. Такое имя нечувствительно к регистру, его можно записывать как строчными, так и прописными буквами. В имени нельзя использовать буквы кириллицы, пробелы, другие специальные символы.

Листинг 1.1. Синтаксис идентификаторов

```
<имя> ::= <буква> | <имя><буква> | <имя><цифра> | <имя>_ | <имя>$

<буква> ::= <прописная буква> | <строчная буква>

<прописная буква> ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P |
Q | R | S | T | U | V | W | X | Y | Z

<строчная буква> ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q
| r | s | t | u | v | w | x | y | z

<цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Следующие имена с точки зрения системы являются одинаковыми:

```
fullname       FULLNAME       FuLlNaMe       FullName
```

Имя с разделителями заключается в кавычки (именно в кавычки, но не в апострофы). Оно может содержать любые символы ASCII, включая буквы кириллицы, пробелы, специальные символы. В нем также могут присутствовать зарезервированные слова. Такое имя является чувствительным к регистру. Имена с разделителем доступны только в диалекте 3.

Синтаксис имени с разделителями:

Листинг 1.2. Синтаксис идентификаторов

```
<имя с разделителями> ::= "<символ ASCII>[<символ ASCII>...]"
```

Следует иметь в виду, что конечные пробелы в именах с разделителями, как и в любых строковых константах, отбрасываются.

Существует определенная похожесть и отличие обычных имен и имен с разделителями. Такие имена с разделителями и обычные, как `"FULLNAME"` и `FULLNAME` являются одинаковыми, а `"FullName"` и `FULLNAME` (так же как, например, и `FullName`) отличаются.

1.8 Константы (литералы)

Литералы служат для непосредственного представления данных. Константа — это значение, подставляемое непосредственно в SQL оператор, которое не получено из выражения, параметра, ссылки на столбец или переменной. Константой может быть строка или число. Ниже приведён список стандартных литералов:

Литералы	Примеры
Целочисленные	`0, -34, 45, 0X080000000`
Вещественные	`0.0, -3.14, 3.23e-23`
Строковые	`'текст', 'don''t!'`
Дата	`DATE '10.01.2014'`
Время	`TIME '15:12:56'`
Временная отметка	`TIMESTAMP '10.01.2014 13:32:02'`
Неопределённое состояние	`null`

1.8.1 Строковые константы

Строковая константа это последовательность символов, заключенных между парой апострофов («одинарных кавычек»). Максимальная длина строковой константы составляет 65535 байт; максимальная количество символов будет определяться количеством байт, используемых для кодирования каждого символа.

Двойные кавычки не должны (допускаются 1 диалектом) использоваться для квотирования строк. В SQL они предусмотрены для других целей.

Если литерал апострофа требуется в строковой константе, то он может быть «экранирован» другим предшествующим апострофом:

```
'Mother O''Reilly''s home-made hooch'
```

Вместо двойного апострофа можно использовать другой символ или пару символов. Ключевое слово `q` или `Q` предшествующее строке в кавычках сообщает парсеру, что некоторые левые и правые

пары одинаковых символов являются разделителями для встроенного строкового литерала.

```
<строковая константа> ::=
                {q|Q}'<символ начала строки> [<символ>...] <символ конца строки>'
```

Символ конца строки должен совпадать с символом начала строки, за исключением символов `'('-')'`, `'{'-'}'`, `'['-']'` и `'<'-'>'`, которые нужно использовать с паре. Внутри строки можно ставить одинарные кавычки.

```
SELECT Q'{abc{def}ghi}' FROM rdb$database;          -- abc{def}ghi
SELECT q'!That's a string!' FROM rdb$database;      -- That's a string
```

Необходимо быть осторожным с длиной строки, если значение должно быть записано в столбец типа `VARCHAR`. Максимальная длина строки для типа `VARCHAR` составляет 32765 байт (32767 для типа `CHAR`). Если значение должно быть записано в столбец типа `BLOB`, то максимальная длина строкового литерала составляет 65535 байт.

Предполагается, что набор символов строковой константы совпадает с набором символов столбца предназначенного для её сохранения.

При необходимости, строковому литералу может предшествовать имя набор символов, который начинается с префикса подчеркивания «_». Это известно как вводный синтаксис (Introducer syntax). Его цель заключается в информировании РЕД Базы Данных о том, как интерпретировать и хранить входящую строку.

```
INSERT INTO People VALUES (_ISO8859_1 'Hans-Jörg Schäfer');
```

Строковые константы могут быть записаны в шестнадцатеричной нотации, так называемые «двоичные строки». Каждая пара шестнадцатеричных цифр определяет один байт в строке. Строки введённые таким образом будут иметь кодировку `OCTETS` по умолчанию, но вводный синтаксис (introducer syntax) может быть использован для принудительной интерпретации строки в другом наборе символов.

Листинг 1.3. Синтаксис двоичных строк

```
{x|X}'<шестнадцатеричная строка>'

<шестнадцатеричная строка> ::= четное количество <шестнадцатеричных цифр>

<шестнадцатеричная цифра> ::= 0..9 | A..F | a..f
```

```
SELECT x'4E657276656E' FROM rdb$database;    -- 4E657276656E  (6-byte 'binary' string)

SELECT _ascii x'4E657276656E' FROM rdb$database;   -- Nerven  (same string, now inter-
                                                              preted as ASCII text)
SELECT _iso8859_1 x'53C3A46765' FROM rdb$database; -- Säge  (4 chars, 4 bytes)

SELECT _utf8 x'53C3A46765' FROM rdb$database;      -- Säge  (4 chars, 5 bytes)
```

Как будут отображены двоичные строки зависит от интерфейса клиента. Например, утилита `isql` использует заглавные буквы `A-F`, в то время как `FlameRobin` буквы в нижнем регистре. Другие могут использовать другие правила конвертирования, например отображать пробелы между парами байт: `'4E 65 72 76 65 6E'`.

Шестнадцатеричная нотация позволяет вставить любой байт (включая 00) в любой позиции в строке.

1.8.2 Числовые константы

Числовая константа — это любое правильное число в одной из поддерживаемых нотаций:

- В SQL, для чисел в стандартной десятичной записи, десятичная точка всегда представлена символом точки и тысячи не разделены. Использование запятых, пробелов, и т.д. вызовет ошибки.
- Экспоненциальная запись, например число `0.0000234` может быть записано как `2.34e-5`. Латинская буква E (равно как и e) означает, что после нее указывается порядок числа — целое число со знаком, задающее степень числа 10. Если целая часть в мантиссе числа отсутствует, то символ нуля, как и в большинстве языков программирования, можно не указывать.
- Запись чисел в двоичном, восьмеричном и шестнадцатеричном формате.
- В числовых литералах можно использовать подчёркивание (`"_"`).

Синтаксис числовых констант:

```
<числовой литерал со знаком> ::=
       [ <знак> ] <беззнаковый числовой литерал>

<беззнаковый числовой литерал> ::=
       <точная запись числа>
     | <экспоненциальная запись>

<точная запись числа> ::=
       <беззнаковое целое число>
     | <беззнаковое десятичное целое число>.[ <беззнаковое десятичное целое число> ]
     | .<беззнаковое десятичное целое число>

<знак> ::=
       <знак плюс>
     | <знак минус>

<экспоненциальная запись> ::= <мантисса> E <экспонента>

<мантисса> ::= <точная запись числа>

<экспонента> ::= <десятичное целое число со знаком>

<десятичное целое число со знаком> ::= [ <знак> ] <беззнаковое десятичное целое число>

<целое число со знаком> ::= [ <знак> ] <беззнаковое целое число>

<беззнаковое целое число> ::=
       <беззнаковое десятичное целое число>
     | <беззнаковое шестнадцатеричное целое число>
     | <беззнаковое восьмеричное целое число>
     | <беззнаковое двоичное целое число>

<беззнаковое десятичное целое число> ::=
       <цифра> [ { [<подчёркивание>] <цифра> }... ]
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
<беззнаковое шестнадцатеричное целое число> ::=
       0X { [<подчёркивание>] <шестнадцатеричная цифра> }...

<беззнаковое восьмеричное целое число> ::=
       0O { [<подчёркивание>] <восьмеричная цифра> }...

<беззнаковое двоичное целое число> ::=
       0B { [<подчёркивание>] <двоичная цифра> }

<знак плюс> ::= + | U+002B

<знак минус> ::= - | U+002D

<точка> ::= . | U+002E

<подчёркивание> ::= _ | U+005F

<шестнадцатеричная цифра> ::=
       0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
     | A | B | C | D | E | F | a | b | c | d | e | f

<десятичная цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<восьмеричная цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7

<двоичная цифра> ::= 0 | 1
```

В числовых литералах допускаются подчёркивания (например, для наглядной записи больших чисел). Символ нижнего подчеркивания игнорируется и число приобретает вид, как если бы запись была без него.

В десятичных числах подчёркивание можно использовать в середине литерала, но не перед точкой и не после неё, например:

```
10_10,
10_10.10_10,
10.10E-10_0
```

В недесятичных числах подчёркивание допускается после литералов, обозначающих систему счисления:

```
0x_FFFF,
0xFF_FF,
0x_FF_FF
```

Подчёркивание в начале и в конце литерала, двойное подчёркивание не допускается ни в каких случаях. Следующие варианты записи приведут к ошибке:

```
0xFF__FF,
0xFFFF_,
_1010,
1010._1010,
```

```
10.10E_-100_
```

Примеры использования подчёркиваний в числовых литералах:

```
SELECT 1234567_890 FROM rdb$database;      -- 1234567890
SELECT -1_234_567_890 FROM rdb$database;   -- -1234567890
SELECT 123_45.67809 FROM rdb$database;     -- 12345.67809
SELECT 123000E-1_0 FROM rdb$database;      -- 1.2300000000000000e-05
SELECT 0o12_34_56_70 FROM rdb$database;    -- 2739128
SELECT -0o_12345670 FROM rdb$database;     -- -2739128
```

1.8.3 Константы даты и времени

Константы даты и времени имеют некоторую специфику. Для представления даты используется множество форматов:

```
dd.mm.yyyy
mm-dd-yyyy
mm/dd/yyyy
yyyy-mm-dd
yyyy/mm/dd
yyyy.mm.dd
dd-MON-yyyy
```

Здесь `MON` — трехсимвольное сокращенное название месяца (английское). Может принимать значения в любом регистре: `jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec` (месяцы с января по декабрь). При описании формата типа данных для указания номера дня в месяце используются символы «`dd`» (число от 1 до 31), для месяца в году — «`mm`» (число от 1 до 12), для номера года — «`yyyy`» (число от 1 до 9999).

Тип данных время позволяет хранить время с точностью до десятитысячной доли секунды (до 100 микросекунд) и задается литералом в виде:

```
hh:mm:ss.nnnn
```

Здесь `hh` — часы: число от 0 до 23, `mm` — минуты: число от 0 до 59, `ss` — секунды: число от 0 до 59, `nnnn` — десятитысячные доли секунды, число от 0000 до 9999. Для часов, минут и секунд ведущий ноль можно не указывать.

Допустимо также использование такого варианта, где вместо двоеточий в качестве разделителей для часов, минут и секунд используются точки:

```
hh.mm.ss.nnnn
```

Более подробно все эти типы данных и операции над ними рассматриваются в главе 3.

1.9 Операторы SQL

SQL операторы включают в себя операторы для сравнения, вычисления, оценки и конкатенации значений.

```
<оператор> ::= <оператор конкатенации>
             | <арифметический оператор>
             | <оператор сравнения>
             | <логический оператор>
<оператор конкатенации> ::= ||

<арифметический оператор> ::= * | / | + | -

<оператор сравнения> ::=
               = | <> | != | ~= | ^= | > | < | >= | <= | !> | ~> | ^> | !< | ~< | ^<

<логический оператор> ::= NOT | AND | OR
```

Все операторы разбиты на 4 типа. Каждый тип оператора имеет свой приоритет. Чем выше приоритет типа оператора, тем раньше он будет вычислен. Внутри одного типа операторы имеют собственный приоритет, который также определяет порядок их вычисления в выражении. Операторы с одинаковым приоритетом вычисляются слева направо. Для изменения порядка вычислений операции могут быть сгруппированы с помощью круглых скобок.

Таблица 1.4 — Приоритеты операторов

Тип оператора	Приоритет	Пояснение
Конкатенация	1	Строки объединяются до выполнения любых других операций.
Арифметический	2	Арифметические операции выполняются после конкатенации строк, но перед выполнением операторов сравнения и логических операций.
Сравнение	3	Операции сравнения вычисляются после конкатенации строк и выполнения арифметических операций, но до логических операций.
Логический	4	Логические операторы выполняются после всех других типов операторов.

Таким образом порядок выполнения операций следующий:

1. Действия в скобках.
2. Операции конкатенации.
3. Операции умножения и деления.
4. Операции сложения и вычитания.
5. Операции сравнения.
6. Операторы `IN, BETWEEN, LIKE, CONTAINING, STARTING WITH, IS NULL, IS DISTINCT FROM`, функции `EXISTS, SINGULAR`, встроенные функции, UDF.
7. Логическое отрицание.
8. Конъюнкция.
9. Дизъюнкция.

1.9.1 Оператор конкатенации

Оператор конкатенации `||` соединяет две символьные строки и создаёт одну строку. Символьные стоки могут быть константами или значениями, полученными из столбцов или других выражений.

```
SELECT LAST_NAME || ', ' || FIRST_NAME AS FULL_NAME FROM EMPLOYEE;
```

1.9.2 Арифметические операторы

Таблица 1.5 — Приоритет арифметических операторов

Оператор	Название	Приоритет
+	Унарный плюс	1
-	Унарный минус	1
*	Умножение	2
/	Деление	2
+	Сложение	3
-	Вычитание	3

```
UPDATE T SET A = 4 + 1/(B-C)*D
```

1.9.3 Оператор сравнения

Имеется шесть традиционных операторов сравнения:

Листинг 1.4. Операторы сравнения

```
<оператор сравнения> ::=   = , < , > , <= , >= , !< , !> , <> , != , ^= , ^> , ^<
```

В операторе сравнения символы «!» и «^» означают отрицание. Оператор может быть применен к любому типу данных столбцов таблицы, за исключением типа данных BLOB. Допустимо сравнение однотипных или близких типов данных. При необходимости можно выполнить явное преобразование типа у операндов сравнения, используя функцию CAST. Список операторов сравнения и их значение приведены в таблице 1.6:

Таблица 1.6 — Приоритет операторов сравнения

Оператор	Название	Приоритет
=	Равно, идентично	2
<> , != , ^=, ~=	Не равно	2
>	Больше	2
<	Меньше	2
>=, !<, ^<, ~<	Больше или равно, не меньше	2
<=, !>, ^>, ~>	Меньше или равно, не больше	2

Результатом сравнения, когда один из операндов или оба имеют значение NULL, всегда будет UNKNOWN, то есть условие не выполняется.

Символьный тип данных можно сравнивать с любым типом данных, кроме BLOB. В таких операциях сравнения осуществляется неявное преобразование других типов данных к символьному. Лучшим же вариантом в сравнении является явное преобразование с использованием функции CAST сравниваемых типов данных к символьному типу.

Сравнение числовых данных между собой никогда не вызывает исключений. Например, можно сравнивать целочисленный тип данных с числом с фиксированной или с плавающей точкой.

Недопустимо сравнение даты или времени с числом или с символьным данным, содержащим строку, не являющуюся датой или временем (иногда это не приведет к выдаче синтаксической ошибки, но на практике не является разумным решением). Дату или время можно сравнивать с символьным данным, если строка содержит дату или время в «правильном» виде; при этом лучше всего следует выполнить явное преобразование строки к нужному типу, использовав функцию `CAST`.

Нельзя дату сравнивать со временем.

```
SELECT *
FROM Pc
WHERE speed >= 500 AND price < 800;

SELECT *
FROM Printer
WHERE type = 'matrix' AND price < 300;
```

Оператор равенства

Оператор `"="`, который используется во многих условиях, сравнивает только значения со значениями. В соответствии со стандартом SQL, `NULL` не является значением и, следовательно, два значения `NULL` не равны и ни неравны друг с другом. Если необходимо, чтобы значения `NULL` соответствовали друг другу при объединении, используйте оператор `IS NOT DISTINCT FROM`. Этот оператор возвращает истину, если операнды имеют то же значение, или, если оба они равны `NULL`.

```
SELECT * FROM A
JOIN B ON A.id IS NOT DISTINCT FROM B.code;
```

Точно так же, если вы хотите чтобы значения `NULL` отличались от любого значения и два значения `NULL` считались равными, используйте оператор `IS DISTINCT FROM` вместо оператора `"<>"`.

```
SELECT * FROM A
JOIN B ON A.id IS DISTINCT FROM B.code;
```

1.9.4 Логические операторы

В SQL используется не обычная двухзначная, а трехзначная логика. В ней присутствует не два, а три значения — `TRUE` (истина), `FALSE` (ложь) и `UNKNOWN` (неопределенное или неизвестное значение).

Таблица 1.7 — Приоритет логических операторов

Оператор	Название	Приоритет
`NOT`	Отрицание условия поиска.	1
`AND`	Объединяет два предиката и более, каждый из которых должен быть истинным, чтобы истинным был и весь предикат.	2

(разрыв таблицы)

(разрыв таблицы)

Оператор	Название	Приоритет
OR	Объединяет два предиката и более, из которых должен быть истинным хотя бы один предикат, чтобы истинным был и весь предикат.	3

В операции отрицания присутствует один операнд. Результат выполнения отрицания представлен в таблице.

Таблица 1.8 — Операция отрицания NOT

Операнд	NOT операнд
TRUE	FALSE
FALSE	TRUE
UNKNOWN	UNKNOWN

В операции дизъюнкции (логическое ИЛИ, OR) участвуют два операнда. Операция симметрична. Результат выполнения дизъюнкции показан в таблице.

Таблица 1.9 — Операция дизъюнкции OR

Операнд 1	Операнд 2	Операнд 1 OR Операнд 2
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	FALSE	FALSE
TRUE	UNKNOWN	TRUE
FALSE	UNKNOWN	UNKNOWN
UNKNOWN	UNKNOWN	UNKNOWN

В операции конъюнкции (логическое И, AND) участвуют два операнда. Операция симметрична. Результат выполнения конъюнкции показан в таблице.

Таблица 1.10 — Операция дизъюнкции AND

Операнд 1	Операнд 2	Операнд 1 AND Операнд 2
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	FALSE	FALSE
TRUE	UNKNOWN	UNKNOWN
FALSE	UNKNOWN	FALSE
UNKNOWN	UNKNOWN	UNKNOWN

1.10 Специальные символы

Существует набор специальных символов, используемых в качестве разделителей, знаков операций (арифметических, строковых, логических) и др.

```
<специальный символ SQL> ::= <пробел> | " | % | & | ' | ( | ) | * | + | , | - | . | /
| : | ; | < | = | > | ? | [ | ] | ^{} | { | }
```

Кроме этих символов в строковых константах и в именах с разделителями могут присутствовать любые символы, включая буквы кириллицы.

1.11 Комментарии

В SQL скриптах, операторах SQL и PSQL модулях могут встречаться комментарии. Комментарий — это произвольный текст заданный пользователем, предназначенный для пояснения работы отдельных частей программы. Синтаксический анализатор игнорирует текст комментариев. В РЕД Базе Данных поддерживается два типа комментариев: блочные и однострочные.

Листинг 1.5. Синтаксис комментария

```
<комментарий> ::= <блочный комментарий> | <однострочный комментарий>

<блочный комментарий> ::= /*<ASCII символ>[ <ASCII символ> ...]*/

<однострочный комментарий>  ::= --<ASCII символ>[ <ASCII символ> ...] <конец строки>
```

Блочные комментарии начинаются с символов `/*` и заканчиваются символами `*/`. Блочные комментарии могут содержать текст произвольной длины и занимать несколько строк.

Однострочные комментарии начинаются с символов `--` и действуют до конца текущей строки.

```
set term !;
CREATE PROCEDURE P(A INT)
RETURNS (B INT)
AS
BEGIN
  /* Данный текст не будет учитываться
  при работе процедуры, т.к. является комментарием
  */
  B = A + 1; -- Однострочный комментарий
  SUSPEND;
END!
set term ;!
```

Глава 2

Выражения

Выражения SQL представляют формальные методы для вычисления, преобразования и сравнения значений. Выражения SQL могут включать в себя столбцы таблиц, переменные, константы, литералы, различные операторы и предикаты, а так же другие выражения. Полный список допустимых элементов в выражениях описан ниже.

Таблица 2.1 — Описание элементов в выражениях

Элемент	Описание
Имя столбца	Идентификаторы столбцов из указанных таблиц, используемые в вычислениях, или сравнениях, или в качестве условия поиска. Столбец типа массив не может быть элементом выражения, если только он не проверяется на `IS [NOT] NULL`
Элементы массива	В выражении может содержаться ссылка на элемент массива, т.е. `<имя массива>[<индекс>]`
Арифметические операторы	Символы `+, -, *, /` используемые для вычисления значений
Оператор конкатенации	Оператор `||` используется для соединения символьных строк.
Логические операторы	Зарезервированные слова `NOT, AND` и `OR` используются при комбинировании простых условий поиска для создания сложных утверждений
Операторы сравнения	Символы `=, <>, !=, ~=, ^=, <, <=, >, >=, !<, ~<, ^<, !>, ~>` и `^>`
Предикаты сравнения	`LIKE, STARTING WITH, CONTAINING, SIMILAR TO, BETWEEN, IS [NOT] NULL` и `IS [NOT] DISTINCT FROM`
Предикаты существования	Предикаты, используемые для проверки существования значений в наборе. Предикат `IN` может быть использован как с наборами констант, так и со скалярными подзапросами. Предикаты `EXISTS, SINGULAR, ALL ANY, SOME` могут быть использованы только с подзапросами.
Константы	Числа, заключённые в апострофы строковые литералы, логические значения `TRUE, FALSE` и `UNKNOWN`, псевдозначение `NULL`
Литералы дат	Выражения, подобные строковым литералам, заключённые в апострофах, которые могут быть интерпретированы как значения даты, времени или даты-времени. Литералами дат могут быть предварительно объявленные литералы (`'TODAY', 'NOW'` и т.д.) или строки из символов и чисел, такие как `'25.12.2016 15:30:35'`, которые могут быть преобразованы в дату, время или дату с временем
Контекстные переменные	Встроенные контекстные переменные
Подзапросы	Оператор `SELECT` заключённый в круглые скобки, который возвращает одно единственное (скалярное) значение или множество значений (при использовании в предикатах существования)
Локальные переменные	Локальные переменные, входные или выходные параметры PSQL модулей (хранимых процедур, триггеров, анонимных блоков PSQL).

(разрыв таблицы)

(разрыв таблицы)

Элемент	Описание
Позиционные параметры	В DSQL в качестве параметров запроса могут быть использованы только позиционные параметры. Позиционные параметры представляют собой знаки вопроса (?) внутри DSQL оператора. Доступ к таким параметрам осуществляется по его номеру (позиции в запросе относительно предыдущего позиционного параметра) поэтому они называются позиционными. Обычно компоненты доступа позволяют работать с именованными параметрами, которые они сами преобразовывают в позиционные
Идентификаторы функций	Идентификаторы встроенных или внешних функций в функциональных выражениях
Приведения типа	Выражение явного преобразования одного типа данных в другой (`CAST`) или сокращённое (для даты/времени) преобразование типа
Условные выражения	Выражение `CASE` и встроенные функции `COALESCE, NULLIF`
Круглые скобки	Пара скобок (...) используются для группировки выражений. Операции внутри скобок выполняются перед операциями вне скобок. При использовании вложенных скобок, сначала вычисляются значения самых внутренних выражений, а затем вычисления перемещаются наверх по уровням вложенности
Предложение COLLATE	Предложение применяется к типам `CHAR` и `VARCHAR`, чтобы в указанной кодировке установить параметры сортировки, используемые при сравнении
NEXT VALUE FOR	Конструкция `NEXT VALUE FOR` позволяет получить следующее значение последовательности, то же самое делает встроенная функция `GEN_ID()`
Выражение AT	Выражение для изменения часового пояса даты и времени.

2.1 Подзапросы

Подзапрос — это специальный вид выражения, которое фактически является запросом `SELECT` к другой таблице, включённый в спецификацию основного запроса. Подзапросы пишутся как обычные `SELECT` запросы, но должны быть заключены в круглые скобки. Выражения подзапроса используется следующими способами:

- Для задания выходного столбца в списке выбора `SELECT`;
- Для получения значений или условий для предикатов поиска (предложения `WHERE, HAVING`).

2.1.1 Коррелированные подзапросы

Подзапрос может быть коррелированным (соотнесённым). Запрос называется соотнесённым, когда оба, и внутренний, и внешний, запросы взаимозависимы. Это означает, что для обработки каждой записи внутреннего запроса, должна быть получена также запись внешнего запроса, т.е. внутренний запрос всецело зависит от внешнего.

```
SELECT CUST_NO, CONTACT_FIRST,CONTACT_LAST,PHONE_NO
FROM CUSTOMER C
WHERE EXISTS
          (SELECT *
           FROM SALES S
           WHERE C.CUST_NO = S.CUST_NO AND S.ORDER_DATE = DATE '07.02.1994' );
```

2.1.2 Подзапросы возвращающие скалярный результат

Подзапросы, используемые в предикатах поиска, кроме предикатов существования и количественных предикатов, должны возвращать скалярный результат, то есть не более чем один столбец из одной отобранной строки или одно агрегированное значение, в противном случае, произойдёт ошибка времени выполнения (`"Multiple rows in a singleton select..."`).

```
SELECT
     e.first_name,
     e.last_name,
     e.salary
FROM employee e
WHERE
     e.salary = (SELECT MAX(ie.salary)
                 FROM employee ie);
```

2.2 Предикаты

Предикат — это простое выражение, утверждающее некоторый факт. Предикат может быть истинным (`TRUE`), ложным (`FALSE`) и неопределённым (`UNKNOWN`). В SQL ложный и неопределённый результаты трактуются как ложь.

В SQL предикаты проверяют в ограничении `CHECK`, предложении `WHERE`, выражении `CASE`, условии соединения во фразе `ON` для предложений `JOIN`, а также в предложении `HAVING`. В PSQL операторы управления потоком выполнения проверяют предикаты в предложениях `IF, WHILE` и `WHEN`.

Проверяемые условия не всегда являются простыми предикатами. Они могут быть группой предикатов, каждый из которых при вычислении делает вклад в вычислении общей истинности. Такие сложные условия называются *утверждениями*. *Утверждения* могут состоять из одного или нескольких предикатов, связанных логическими операторами `AND, OR` и `NOT`.

Каждый из предикатов может содержать вложенные предикаты. Результат вычисления истинности утверждения получается в результате вычисления всех предикатов по направлению от внутренних к внешним. Каждый «уровень» вычисляется в порядке приоритета, до тех пор, пока невозможно будет получить окончательное утверждение.

2.2.1 Предикаты сравнения

Предикат сравнения представляет собой два выражения, соединяемых оператором сравнения. К операторам сравнения относятся традиционные операторы (`< , > , =, ...`), а также другие перечисленные далее операторы.

Оператор BETWEEN

Оператор `BETWEEN` проверяет, попадает (или не попадает при использовании `NOT`) ли значение во включающий диапазон значений (включая границы).

Листинг 2.1. Синтаксис оператора `BETWEEN`

```
<значение> [NOT] BETWEEN <значение 1> AND <значение 2>
```

Условие будет истинным, если значение присутствует в указанном диапазоне (от `<значение 1>` включительно до `<значение 2>` включительно) при отсутствии ключевого слова `NOT`. При наличии ключевого слова `NOT` условие будет истинным, если значение отсутствует в указанном диапазоне, включая граничные значения.

Оператор BETWEEN является включающим, то есть значения, совпадающие с границами диапазона, дают значение "истина". Чтобы исключить граничные значения из условия, нужно создать несколько более сложную конструкцию, например:

```
(<значение> BETWEEN <значение 1> AND <значение 2>) AND
(<значение> NOT IN (<значение 1>, <значение 2>))
```

Оператор BETWEEN использует два аргумента совместимых типов. В отличие от некоторых других СУБД в РЕД Базе Данных оператор BETWEEN не является симметричным. Меньшее значение должно быть первым аргументом, иначе предикат BETWEEN всегда будет ложным.

Пример

Если в таблице сотрудников STAFF нужно выбрать только тех сотрудников, которые имеют оклады (столбец SALARY) в соответствующем диапазоне, включая граничные значения, то следует выполнить следующий оператор:

```
SELECT EMP_NO, FIRST_NAME, LAST_NAME,PHONE_EXT
FROM EMPLOYEE
WHERE SALARY BETWEEN 100000 AND 200000;
```

Оператор LIKE

Предикат LIKE сравнивает выражение символьного типа с шаблоном.

Листинг 2.2. Синтаксис оператора LIKE

```
<значение> [NOT] LIKE <шаблон> [ESCAPE <символ экранирования>]
```

Этот вариант является чувствительным к регистру (за исключением случаев, когда само поле определено с сортировкой (COLLATION) нечувствительной к регистру).

В шаблоне можно указать трафаретные символы % и _. Символ процента задает произвольное количество, в том числе и нулевое, любых символов. Знак подчеркивания задает ровно один любой символ.

```
SELECT DEPT_NO FROM DEPARTMENT WHERE DEPARTMENT LIKE 'Software%';
```

Необязательное ключевое слово ESCAPE позволяет в шаблон включить и сами трафаретные символы % и _. После ESCAPE нужно указать символ, который должен предшествовать в шаблоне трафаретному символу, когда такой трафаретный символ должен рассматриваться как обычный символ, присутствующий в строке.

```
SELECT
RDB$RELATION_NAME
FROM RDB$RELATIONS
WHERE RDB$RELATION_NAME LIKE '%#_%' ESCAPE '#';
```

Для того чтобы в строке также можно было использовать обычным образом и символ, заданный после ключевого слова ESCAPE, необходимо задать его в этой строке дважды.

В общем случае предикат LIKE не использует индекс. Однако если предикат принимает вид LIKE 'строка%', то он будет преобразован в предикат STARTING WITH, который будет использовать индекс. Если вам необходимо выполнить поиск с начала строки, то вместо предиката LIKE рекомен-

дуется использовать предикат `STARTING WITH`.

Оператор STARTING WITH

Оператор `STARTING WITH` ищет строку или тип, подобный строке, которая начинается с символов в его аргументе.

Листинг 2.3. Синтаксис оператора `STARTING WITH`

```
<значение> [NOT] STARTING [WITH] <значение 1>
```

Оператор чувствителен к регистру, однако такое ограничение также можно легко обойти, используя функцию `UPPER` или функцию `LOWER`.

При использовании предиката `STARTING WITH` в поисковых условиях DML запросов, оптимизатор может использовать индекс по искомому столбцу, если он определён.

```
SELECT LAST_NAME, FIRST_NAME
FROM EMPLOYEE
WHERE LAST_NAME STARTING WITH 'Jo';
```

Оператор CONTAINING

Оператор `CONTAINING` ищет строку или тип, подобный строке, отыскивая последовательность символов, которая соответствует его аргументу.

Листинг 2.4. Синтаксис оператора `CONTAINING`

```
<значение> [NOT] CONTAINING <значение 1>
```

Оператор может быть использован для алфавитно-цифрового (подобного строковому) поиска в числах и датах.

Поиск `CONTAINING` не чувствителен к регистру. Тем не менее, если используется сортировка чувствительная к акцентам, то поиск будет чувствителен к акцентам.

При использовании оператора `CONTAINING` во внимание принимаются все символы строки. Это касается так же начальных и конечных пробелов. Если операция сравнения в запросе должна вернуть все строки, содержащие строки `CONTAINING 'абв '` (с символом пробела на конце), то строка, содержащая `'абв'` (без пробела), не будет возвращена.

При использовании предиката `CONTAINING` в поисковых условиях DML запросов, оптимизатор не может использовать индекс по искомому столбцу.

```
SELECT PROJ_NAME
FROM PROJECT
WHERE PROJ_NAME CONTAINING 'map';

PROJ_NAME
====================
AutoMap
MapBrowser port
```

Оператор SIMILAR TO

Оператор `SIMILAR TO` проверяет соответствие строки шаблону регулярного выражения SQL. Для успешного выполнения шаблон должен соответствовать всей строке — соответствие подстроки не достаточно. Если один из операндов имеет значение `NULL`, то и результат будет `NULL`. В противном случае результат является `TRUE` или `FALSE`. Предикат может быть использован в любом контексте, который принимает булевы (логические) выражения, такие как предложения `WHERE`, ограничения `CHECK` и PSQL-оператор `IF()`.

Синтаксис оператора `SIMILAR TO`:

Листинг 2.5. Синтаксис оператора `SIMILAR TO`

```
<строка> [ NOT ] SIMILAR TO <регулярное выражение> [ESCAPE <символ
экранирования>]

<регулярное выражение> := <строковый элемент> [<квантификатор>] [['|'] <строковый
элемент> [ <квантификатор>] ...]

<квантификатор> ::= ? | * | + | '{' m [,[n]] '}'

<строковый элемент> ::= { <экранированный символ> | <обычный символ>}
                        | %
                        | <класс символов>
                        | ( <регулярное выражение> )

<экранированный символ> ::= <символ экранирования> <специальный символ>
                          | <символ экранирования> <символ экранирования>

<специальный символ> ::= '[' | ']' | '(' | ')' | '|' | '^' | '-' | '+' | '*' | '%' |
'_' | '?' | '{' | '}'

<обычный символ> ::= любой символ за исключением <специальный символ> и не
эквивалентный <символ экранирования> (если задан)

<класс символов> ::= '_'
                   | '[' <элемент класса> ... ']'
                   | '[^' <не элемент класса> ... ']'
                   | '[' <элемент класса> ... '^' <не элемент класса> ... ']'

<элемент класса>, <не элемент класса> ::= {<экранированный символ>|<обычный символ>}
                                        | <диапазон>
                                        | <предопределенные классы>

<диапазон> ::= { <экранированный символ> | <обычный символ>} -
               { <экранированный символ> | <обычный символ>}

<предопределенные классы> ::= '[:'ALPHA':]' | '[:'UPPER':]' | '[:'LOWER':]'
                            | '[:'DIGIT':]' | '[:'ALNUM':]' | '[:'SPACE':]'
                            | '[:'WHITESPACE':]'
```

Символы

Регулярные выражения в основном состоят из символов, представляющих сами себя. Но есть исключения - это специальные символы:

```
[] ( ) | ^ - + * % _ ? { }
```

и управляющие символы.

Если регулярное выражение не содержит специальных и управляющих символов, то оно соответствует идентичной строке (в зависимости от используемой сортировки).

```
'Symbol' SIMILAR TO 'Symbol',     -- <true>
'Symbols' SIMILAR TO 'Symbol',    -- <false>
'SYMBOL' SIMILAR TO 'Symbol'      -- в зависимости от сортировки
```

Шаблоны

SQL шаблонам _ и % соответствует любой единственный символ и строка любой длины (в том числе и пустая), соответственно:

```
'Template' SIMILAR TO 'Te_plate',    -- <true>
'Template' SIMILAR TO 'T_plate',     -- <false>
'Template' SIMILAR TO 'T%te'         -- <true>
```

Классы символов

Символы, заключенные в квадратные скобки, определяют класс символов. Если символ в строке соответствует классу, то символ является элементом класса. Причем классу соответствует единственный символ строки.

Два символа, соединенные дефисом, в определении класса определяют диапазон. Диапазон для сопоставления включает в себя эти два конечных символа и все символы, находящиеся между ними.

```
'Class' SIMILAR TO 'Cla[o-y]s',      -- <true>
'Class' SIMILAR TO 'C[la]ss',        -- <false>
'Class' SIMILAR TO 'C[abd-sx]ass'    -- <true>
```

Если определение класса запускается со знаком вставки (^), то все, что следует за ним, исключается из класса. Все остальные символы проверяются. Если знак вставки (^) находится не в начале последовательности, то класс включает в себя все символы до него и исключает символы после него.

```
'Error' SIMILAR TO 'Er[^a-g]or',        -- <true>
'Error' SIMILAR TO 'Err[^e-p][^a-g]',   -- <false>
'Error' SIMILAR TO 'Er[wrt^a-d]or'      -- <true>
```

В определении класса также могут использоваться предопределенные классы символов из таблицы 2.2.

Таблица 2.2 — Идентификаторы класса символов

Идентификатор	Описание
ALPHA	Все латинские буквы (a-z, A-Z)
UPPER	Все прописные латинские буквы (A-Z)
LOWER	Все строчные латинские буквы (a-z)
DIGIT	Все арабские цифры (0-9)
SPACE	Пробел (ASCII 32)

(разрыв таблицы)

(разрыв таблицы)

Идентификатор	Описание
WHITESPACE	Все символы-разделители (горизонтальная табуляция, перевод строки, вертикальная табуляция, возврат каретки, перевод страницы, пробел)
ALNUM	Все латинские буквы (ALPHA) и арабские цифры (DIGIT)

Включение в оператор SIMILAR TO предопределенного класса имеет тот же эффект, как и включение всех его элементов. Использование предопределенных классов допускается только в пределах определения класса. Если определение класса запускается со знаком вставки (^), то всё, что следует за ним, исключается из класса. Все остальные символы проверяются. Если знак вставки (^) находится не в начале последовательности, то класс включает в себя все символы до него и исключает символы после него.

```
'Identifier' SIMILAR TO 'Id[[:ALNUM:]]nti[a-m]ier',     -- <true>
'Identifier' SIMILAR TO 'Ide[[:ALPHA:]^f-o]tifier',     -- <false>
'Identifier' SIMILAR TO 'Ident[^[:DIGIT:]]fier'         -- <true>
```

Кванторы

Квантор после символа, символьного класса или группы определяет, сколько раз предшествующее выражение может встречаться.

Вопросительный знак (?) сразу после символа, класса или группы указывает на то, что для соответствия предыдущий элемент может произойти 0 или 1 раз.

Звёздочка (*) сразу после символа, класса или группы указывает на то, что для соответствия предыдущий элемент может произойти 0 или более раз.

Знак плюс (+) сразу после символа, класса или группы указывает на то, что для соответствия предыдущий элемент может произойти 1 или более раз.

```
'Question' SIMILAR TO 'Questt?ion',       -- <true>
'Asterisk' SIMILAR TO 'Ast[c-s]*sk',      -- <true>
'Plus' SIMILAR TO 'Plus[[:DIGIT:]]+'      -- <false>
```

Если символ или класс сопровождаются числом, заключённым в фигурные скобки ({n}), то для соответствия нужно повторение элемента точно это число раз. Если число сопровождается запятой ({n,}), то для соответствия нужно повторение элемента как минимум это число раз. Если фигурные скобки содержат два числа ({m,n}) и второе число больше первого, то для соответствия элемент должен быть повторен как минимум m раз и не больше n раз.

```
'Braces' SIMILAR TO 'Bra{2}ces',        -- <false>
'Braces' SIMILAR TO 'Bra[aceg]{2,}s',   -- <true>
'Braces' SIMILAR TO 'Br[aceg]{1,2}s'    -- <false>
```

Условие ИЛИ

В условиях регулярных выражений можно использовать оператор ИЛИ (|). Соответствие произошло, если строка параметра соответствует по крайней мере одному из условий:

```
'Condition' SIMILAR TO 'Condi|tion',                -- <false>
'Condition' SIMILAR TO 'Condition|Statement',       -- <true>
'Condition' SIMILAR TO 'Condi_+|Kondi_+|Ckondi_+'   -- <true>
```

Группы

Одна или более частей регулярного выражения могут быть сгруппированы в подвыражения. Для этого их нужно заключить в круглые скобки:

```
'Groups' SIMILAR TO 'G(ru|ro|ra)ups'     -- <true>
```

Экранирование специальных символов

Для исключения из процесса сопоставления специальных символов (которые часто встречаются в регулярных выражениях) их надо экранировать. Специальных символов экранирования по умолчанию нет — их при необходимости определяет пользователь:

```
'Russia(RU)' SIMILAR TO 'R[a-z]+\(R[A-Z]+\)' ESCAPE '\',      -- <true>
'France[FR]' SIMILAR TO 'Fr[a-z]+#[F%#]' ESCAPE '#',     -- <true>
'Puerto-Rico' SIMILAR TO 'P%$-R%' ESCAPE '$'              -- <true>
```

Оператор IS DISTINCT FROM

Это оператор проверки на неравенство (равенство, если задано NOT) двух значений.

Листинг 2.6. Синтаксис оператора IS DISTINCT FROM

```
<значение 1> IS [NOT] DISTINCT FROM <значение 2>
```

Два операнда считают различными (DISTINCT), если они имеют различные значения, или если одно из них — NULL, и другое нет. Они считаются равными (NOT DISTINCT), если имеют одинаковые значения или оба имеют значение NULL.

```
SELECT FISCAL_YEAR, PROJ_ID, PROJECTED_BUDGET
FROM PROJ_DEPT_BUDGET
WHERE FISCAL_YEAR IS DISTINCT FROM '1994';
```

В отличие от операторов равно (=) и не равно (!=, <>) этот оператор трактует два сравниваемых пустых значения NULL как равные друг другу. Как и в случае оператора IS [NOT] NULL данный оператор всегда возвращает либо TRUE, либо FALSE.

Таблица 2.3 — Результаты выполнения различных операторов сравнения

	=	IS NOT DISTINCT FROM	!=, <>	IS DISTINCT FROM
Одинаковые значения	TRUE	TRUE	FALSE	FALSE
Различные значения	FALSE	FALSE	TRUE	TRUE
Оба NULL	UNKNOWN	TRUE	UNKNOWN	FALSE
Одно NULL	UNKNOWN	FALSE	UNKNOWN	TRUE

Оператор IS

Оператор IS проверяет, что выражение в левой части является псевдо значением NULL или соответствует логическому значению в правой части.

Листинг 2.7. Синтаксис оператора IS

```
<значение> IS [NOT] {TRUE | FALSE | UNKNOWN | NULL}
```

Если в правой части предиката использованы литерал `TRUE, FALSE` или `UNKNOWN`, то выражение в левой части должно быть логического типа, иначе будет выдана ошибка.

Оператор может вернуть только истинное значение `TRUE` или ложное `FALSE`, значение `UNKNOWN` невозможно.

2.2.2 Предикаты существования

В эту группу предикатов включены предикаты, которые используют подзапросы и передают значения для всех видов утверждений в условиях поиска. Предикаты существования называются так потому, что они различными способами проверяют существование или отсутствие результатов подзапросов.

Предикат EXISTS

Если результат подзапроса будет содержать хотя бы одну запись, то предикат `EXISTS` оценивается как истинный (`TRUE`), в противном случае предикат оценивается как ложный (`FALSE`).

Листинг 2.8. Синтаксис предиката `EXISTS`

```
[NOT] EXISTS (<оператор SELECT>)
```

Аргументом предиката `EXISTS` является оператор `SELECT`, возвращающий произвольное количество любых столбцов таблицы.

```
SELECT *
FROM employee
WHERE NOT EXISTS (SELECT *
                  FROM employee_project ep
                  WHERE ep.emp_no = employee.emp_no);
```

Предикат IN

Предикат `IN` проверяет, присутствует ли значение выражение в указанном справа наборе значений.

Листинг 2.9. Синтаксис предиката `IN`

```
<значение> [NOT] IN (<оператор SELECT> | <список значений>)

<список значений> ::= <значение_1> [, <значение_2> ...]
```

Набор значений не может превышать 1500 элементов. Предикат `IN` может быть переписан в следующей эквивалентной форме:

```
(<значение> = <значение_1> [OR <значение> = <значение_2> ...])
```

```
SELECT *
FROM EMPLOYEE
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
WHERE FIRST_NAME IN ('Pete', 'Ann', 'Roger');
```

Во второй форме предикат IN проверяет, присутствует (или отсутствует, при использовании NOT IN) ли значение выражения слева в результате выполнения подзапроса справа. Результат подзапроса может содержать только один столбец, иначе будет выдана ошибка "count of column list and variable list do not match".

Запросы с использованием предиката IN с подзапросом, можно переписать на аналогичный запрос с использованием предиката EXISTS.

Например, следующий запрос:

```
SELECT model, speed, hd
FROM PC
WHERE model IN (SELECT model
                FROM product
                WHERE maker = 'A');
```

можно переписать на аналогичный запрос с использованием предиката EXISTS:

```
SELECT model, speed, hd
FROM PC
WHERE EXISTS (SELECT *
              FROM product
              WHERE maker = 'A' AND product.model = PC.model);
```

Однако, запрос с использованием NOT IN не всегда даст тот же результат, что запрос NOT EXISTS. Причина заключается в том, что предикат EXISTS всегда возвращает TRUE или FALSE, тогда как предикат IN может вернуть NULL в следующих случаях:

1. Когда проверяемое значение равно NULL и список в IN не пуст.
2. Когда проверяемое значение не имеет совпадений в списке IN и одно из значений является NULL.

В этих двух случаях предикат IN вернёт NULL, в то время как соответствующий предикат EXISTS вернёт FALSE. В поисковых условиях или операторе IF оба результата обозначают «провал» и обрабатываются одинаково.

Однако на тех же данных NOT IN вернёт NULL, в то время как NOT EXISTS вернёт TRUE, что приведёт к противоположному результату.

Символьные данные в предикате чувствительны к регистру.

Этот предикат может применяться к любому типу данных, кроме типа данных BLOB.

В предикате неявно допускается и пустое значение NULL.

При использовании предиката IN в поисковых условиях DML запросов, оптимизатор может использовать индекс по искомому столбцу, если он определён.

Предикат SINGULAR

Предикат SINGULAR использует подзапрос в качестве аргумента и оценивает его как истинный, если подзапрос возвращает одну и только одну строку результата, в противном случае предикат оценивается как ложный.

Листинг 2.10. Синтаксис предиката SINGULAR

```
[NOT] SINGULAR (<оператор SELECT>)
```

Аргументов предиката SINGULAR является оператор SELECT, возвращающий произвольное количество любых столбцов таблицы (обычно это *). Данный предикат может принимать только два значения: истина (TRUE) и ложь (FALSE)

2.2.3 Количественные предикаты подзапросов

Квантором называется логический оператор, задающий количество объектов, для которых данное утверждение истинно. Это логическое количество, а не числовое; оно связывает утверждение с полным множеством возможных объектов. Такие предикаты основаны на формальных логических квантификаторах общности и существования, которые в формальной логике записываются как ∀ и ∃.

Оператор ALL

При использовании квантора ALL, предикат является истинным, если каждое значение выбранное подзапросом удовлетворяет условию в предикате внешнего запроса.

Листинг 2.11. Синтаксис квантора ALL

```
<значение> <оператор сравнения> ALL (<оператор SELECT>)
```

Здесь используется операторы сравнения, описанные выше в этом разделе, а также оператор IS [NOT] DISTINCT FROM.

Если подзапрос не возвращает ни одной строки, то предикат автоматически считается верным.

```
SELECT EMP_NO
FROM EMPLOYEE
WHERE salary > ALL (SELECT salary
                    FROM EMPLOYEE
                    WHERE JOB_COUNTRY = 'France');
```

Операторы SOME и ANY

Синтаксис использования этих кванторов:

Листинг 2.12. Синтаксис кванторов SOME и ANY

```
<значение> <оператор сравнения> { SOME | ANY } (<оператор SELECT>)
```

Ключевые слова SOME и ANY являются синонимами. Результатом будет "истина", если сравнение истинно хотя бы для одного значения, полученного из оператора SELECT.

Если подзапрос не возвращает ни одной строки, то предикат автоматически считается ложным.

Глава 3

Типы данных РЕД Базы Данных

3.1 Перечень типов данных

Тип данных определяет множество значений, которые может содержать столбец таблицы, переменная или параметр хранимой процедуры или триггера. Тип данных определяет также допустимые для соответствующего столбца (переменной, параметра) операции.

Типы данных, используемые в СУБД РЕД База Данных, в алфавитном порядке представлены в таблице 3.1. Далее в этой главе они рассматриваются более подробно. Перечисленные типы данных могут применяться не только при описании доменов, но также и при описании характеристик столбцов таблиц базы данных, входных и выходных параметров хранимых процедур, внутренних переменных хранимых процедур и триггеров (подробнее об этом см. в главе 9 и в главе 17).

Таблица 3.1 — Типы данных, используемые в РЕД Базе Данных

Тип данных	Размер (байт)	Описание
`BIGINT`	8	Числовой тип данных. Хранит целые числа в диапазоне от -2^{63} до $+2^{63}-1$, или от –9,223,372,036,854,775,808 до +9,223,372,036,854,775,807.
`BOOLEAN`	1	Логический тип данных. Может принимать значения `TRUE`, `FALSE` и `UNKNOWN`.
`BINARY(n)`	n	Бинарный тип данных фиксированной длины. Является псевдонимом типа `CHAR(n) CHARACTER SET OCTETS`. Диапазон от 1 до 32 767 байт
`BLOB`	Переменный	Большой двоичный объект (`Binary Large OBject`). Позволяет хранить произвольные данные: форматированные тексты, графику, звуки, видеофильмы. Размер сегмента `BLOB` ограничивается 64K. Максимальный размер поля `BLOB` 4 Гб. Для размера страницы 4096 максимальный размер `BLOB` поля несколько ниже 2 Гб.
`CHAR(n)`, `CHARACTER(n)`	n символов	Символьный тип данных фиксированной длины. Число `n` задает максимальное количество символов. Конечные пробелы в базе данных не хранятся, а восстанавливаются до указанного размера при отображении такого столбца. Восстановление пробельных символов до максимальной длины происходит на клиенте, а не на сервере, при передаче данных по локальной сети пробелы не передаются, что позволяет уменьшить сетевой трафик. Максимальный размер столбца 32767 байтов. Максимальное количество хранимых символов зависит от используемого для столбца набора символов. Некоторые наборы символов для хранения каждого символа используют более одного байта (см. Приложение Д). Если количество символов `n` не указано, принимается 1. Для этого типа данных допустимо сокращение `CHAR`.
`DATE`	4	Даты в диапазоне от 1 января 1 г. до 31 декабря 9999 г.

(разрыв таблицы)

(разрыв таблицы)

Тип данных	Размер (байт)	Описание
`DECIMAL(n, m)`	2, 4, 8 или 16	Числовой тип данных. Число с фиксированной точкой (целое или дробное число), `n` — общее количество знаков в числе, включая дробные знаки (диапазон значений от 1 до 38), `m` — количество знаков после десятичной точки (значения от 0 до 38). Основное требование к параметрам: `m <= n`. Если значения для `n` и/или `m` не указаны, то предполагается (9, 0). Для этого типа данных допустимо сокращение `DEC`.
`DECFLOAT (16|34)`	8 или 16	SQL:2016 совместимый тип данных точно хранящий десятичные числа с плавающей запятой, основанный на стандарте IEEE 754-2008. Количество значащих цифр (точность) - 16 или 34. Если точность не указана, по умолчанию используется 34.
`DOUBLE PRECISION`	8	Числовой тип данных. Число с плавающей точкой. Находится в диапазоне от 2.225×10^{-308} до 1.797×10^{308}. Позволяет хранить до 15 значащих цифр.
`FLOAT`	4	Числовой тип данных. Число с плавающей точкой. Диапазон хранимых чисел от 1.175×10^{-38} до 3.402×10^{38}. Позволяет хранить до 7 значащих цифр.
`INTEGER`	4	Числовой тип данных. Целое число в диапазоне от -2^{31} до $+2^{31}-1$, или от –2,147,483,648 до +2,147,483,647. Для этого типа данных допустимо сокращение INT.
`INT128`	16	Числовой тип данных. Целое число в диапазоне от -2^{127} до $+2^{127}-1$
`NATIONAL CHARACTER (n)`	n символов	Символьный тип данных фиксированной длины. Отличается от типа данных `CHARACTER` только тем, что для него предопределен набор символов `ISO8859_1`. Другие наборы символов для этого типа данных задавать нельзя. Если количество символов `n` не указано, принимается 1. Допустимы следующие сокращения `NATIONAL CHAR, NCHAR`.
`NATIONAL CHARACTER VARYING (n)`	n символов	Символьный тип данных переменной длины. Отличается от типа данных `VARYING CHARACTER` только тем, что для него предопределен набор символов `ISO8859_1`. Другие наборы символов для этого типа данных задавать нельзя. Количество символов `n` обязательно должно быть указано. Допустимы следующие сокращения: `NATIONAL CHAR VARYING, NCHAR VARYING`.
`NUMERIC(n, m)`	2, 4, 8 или 16	Числовой тип данных. Дробное или целое число с фиксированной точкой, `n` — общее количество знаков в числе от 1 до 38, `m` — количество знаков после десятичной точки (число от 0 до 38). Общее требование: `m <= n`. Если `n` и/или `m` не указаны, то предполагается (9, 0). В диалекте 3 полностью соответствует типу данных `DECIMAL`. Сокращение названия для этого типа данных не используется.
`SMALLINT`	2	Целочисленный тип данных. Целое число в диапазоне от -2^{15} до $+2^{15}-1$, или от –32,768 до +32,767.

(разрыв таблицы)

(разрыв таблицы)

Тип данных	Размер (байт)	Описание
TIME [WITHOUT TIME ZONE]	4	Задает время в часах, минутах, секундах и десятитысячных долях секунды без учёта часового пояса. Диапазон: от 00:00:00.0000 до 23:59:59.9999.
TIME WITH TIME ZONE	6	Задает время в часах, минутах, секундах и десятитысячных долях секунды с учётом часового пояса. Диапазон: от 00:00:00.0000 до 23:59:59.9999.
TIMESTAMP [WITHOUT TIME ZONE]	8	Комбинация (объединение) даты и времени без учёта часового пояса. Диапазон: 01-01-0001 AD до 31-12-9999 AD.
TIMESTAMP WITH TIME ZONE	10	Комбинация (объединение) даты и времени с учётом часового пояса. Диапазон: 01-01-0001 AD до 31-12-9999 AD.
VARBINARY(n), BINARY VARYING(n)	n	Бинарный тип данных переменной длины. Является псевдонимом типа VARCHAR(n) CHARACTER SET OCTETS. Диапазон от 1 до 32765 байтов.
VARCHAR(n), CHAR VARYING(n), CHARACTER VARYING(n)	n символов	Символьный тип данных переменной длины. Максимальный размер 32765 байтов. Количество хранимых символов зависит от используемого для столбца набора символов (см. Приложение Д), где для каждого набора символов указано и количество байтов, необходимых для хранения одного символа). Если количество символов n не указано, то предполагается 1024.

Типы данных можно объединить в группы, или категории — числовые данные, строковые, данные даты и времени, двоичные данные и логические данные.

Любой тип данных может быть также массивом произвольной размерности.

3.2 SMALLINT

Тип данных SMALLINT представляет собой 16-битное целое.

Числа типа SMALLINT находятся в диапазоне $-2^{15}..2^{15}-1$, или $-32768..32767$.

Числа SMALLINT не могут быть заданы в шестнадцатеричном виде явно. Но РЕД База Данных будет прозрачно преобразовывать шестнадцатеричное число в SMALLINT, если это необходимо, при условии что оно попадает в допустимый диапазон положительных и отрицательных значений для SMALLINT.

Операции сложения и вычитания для типа данных SMALLINT выполняются обычным образом. Следует только быть особенно внимательными при выполнении операций умножения и деления. В этих операциях результат будет иметь количество дробных знаков, равное сумме дробных знаков обоих операндов. То есть результатом операции деления двух целых чисел:

```
select cast(1 as SMALLINT)/cast(3 as SMALLINT) from rdb$database;
```

будет 0, потому что сумма дробных знаков операндов равна нулю, а целая часть в результате выполнения операции деления будет равна нулю. Результат является целочисленным.

Для числовых типов данных могут использоваться агрегатные функции, определенные в SQL — MIN, MAX, SUM, AVG и множество других встроенных в SQL функций. Описание встроенных функций см. Приложение З.

3.3 INTEGER

Тип данных `INTEGER` представляет собой 32-битное целое. Сокращённый вариант записи типа данных `INT`. Числа типа `INTEGER` находятся в диапазоне $-2^{31}..2^{31}-1$, или −2147483648..2147483647.

Числа типа `INTEGER` могут быть заданы в шестнадцатеричном виде с 1 — 8 шестнадцатеричными цифрами.

Операции сложения и вычитания для типа данных `INTEGER` выполняются обычным образом. Следует только быть особенно внимательными при выполнении операций умножения и деления. В этих операциях результат будет иметь количество дробных знаков, равное сумме дробных знаков обоих операндов. То есть результатом операции деления двух целых чисел:

```
select cast(1 as INTEGER)/cast(3 as INTEGER) from rdb$database;
```

будет 0, потому что сумма дробных знаков операндов равна нулю, а целая часть в результате выполнения операции деления будет равна нулю. Результат является целочисленным.

Для числовых типов данных могут использоваться агрегатные функции, определенные в SQL — `MIN, MAX, SUM, AVG` и множество других встроенных в SQL функций. Описание встроенных функций см. в Приложение 3.

3.4 BIGINT

Тип данных `BIGINT` представляет собой 64-битное целое, совместимый со стандартом SQL-99. Он доступен только в 3-м диалекте. При использовании клиентом диалекта 1, передаваемое сервером значение генератора усекается до 32-х битного целого (`INTEGER`).

Числа типа `BIGINT` находятся в диапазоне $-2^{63}..2^{63}-1$, или −9223372036854775808..9223372036854775807.

Числа типа `BIGINT` могут быть заданы в шестнадцатеричном виде с 9 — 16 шестнадцатеричными цифрами.

Операции сложения и вычитания для типа данных `BIGINT` выполняются обычным образом. Следует только быть особенно внимательными при выполнении операций умножения и деления. В этих операциях результат будет иметь количество дробных знаков, равное сумме дробных знаков обоих операндов. То есть результатом операции деления двух целых чисел:

```
select cast(1 as BIGINT)/cast(3 as BIGINT) from rdb$database;
```

будет 0, потому что сумма дробных знаков операндов равна нулю, а целая часть в результате выполнения операции деления будет равна нулю. Результат является целочисленным.

Для числовых типов данных могут использоваться агрегатные функции, определенные в SQL — `MIN, MAX, SUM, AVG` и множество других встроенных в SQL функций. Описание встроенных функций см. в Приложение 3.

3.5 INT128

Тип данных `INT128` представляет собой 128-битное целое. Числа типа `INT128` находятся в диапазоне $-2^{127}..2^{127}-1$.

Числа типа `INT128` могут быть заданы в шестнадцатеричном виде с 17 — 32 шестнадцатеричными цифрами.

Операции сложения и вычитания для типа данных `INT128` выполняются обычным образом. Следует только быть особенно внимательными при выполнении операций умножения и деления. В этих операциях результат будет иметь количество дробных знаков, равное сумме дробных знаков обоих операндов. То есть результатом операции деления двух целых чисел:

```
select cast(1 as INT128)/cast(3 as INT128) from rdb$database;
```

будет 0, потому что сумма дробных знаков операндов равна нулю, а целая часть в результате выполнения операции деления будет равна нулю. Результат является целочисленным.

Для числовых типов данных могут использоваться агрегатные функции, определенные в SQL — `MIN, MAX, SUM, AVG` и множество других встроенных в SQL функций. Описание встроенных функций см. в Приложение 3.

3.6 Шестнадцатеричный формат для целых чисел

Целочисленные значения могут быть записаны в шестнадцатеричной системе счисления. Шестнадцатеричные числовые константы указываются в формате:

```
<0x><шестнадцатеричное число>
```

где `'0'` и `'X'` (или `'x'`) являются литералом, за которым следует набор байт, используя `0-9`, `a-f` или `A-F`.

Числа состоящие из 1-8 шестнадцатеричных цифр будут интерпретированы как `INTEGER`, состоящие из 9-16 цифр — как `BIGINT`, состоящие из 17-32 цифр — как `INT128`.

```
SELECT 0x6FAA0D3 FROM rdb$database;              -- 117088467
SELECT 0x4F9 FROM rdb$database;                 -- 1273
SELECT 0x6E44F9A8 FROM rdb$database;            -- 1850014120
SELECT -0x9E44F9A8 FROM rdb$database;           -- -2655320488  (BIGINT)
SELECT 0x09E44F9A8 FROM rdb$database;           -- 2655320488  (BIGINT)
SELECT 0x28ED678A4C987 FROM rdb$database;       -- 720001751632263
SELECT -0xFFFFFFFFFFFFFFFF FROM rdb$database;   -- -18446744073709551615
```

Шестнадцатеричные числа в диапазоне `0x0..0x7FFF_FFFF` являются положительными числами типа `INTEGER` со значениями `0..2147483647`.

Шестнадцатеричные числа в диапазоне `0x8000_0000..0x7FFF_FFFF_FFFF_FFFF` являются положительными числами типа `BIGINT` со значениями `2147483648..9223372036854775807`.

Шестнадцатеричные числа в диапазоне от `0x8000_0000_0000_0000` до `0x7FFF_FFFF_FFFF_FFFF_FFFF_FFFF_FFFF_FFFF` являются положительными числами типа `INT128` со значениями `9223372036854775808..170141183460469231731687303715884105727`.

Для представления отрицательных значений нужно поставить знак минус перед беззнаковым шестнадцатиричным литералом.

Числа с типом `SMALLINT` не могут быть записаны в шестнадцатеричном виде, строго говоря, так как даже `0x1` оценивается как `INTEGER`. Тем не менее, если вы записываете положительное целое число в пределах 16-разрядного диапазона от `0x0000` (десятичный ноль) до `0x7FFF` (десятичное 32767), то оно будет преобразовано в `SMALLINT` прозрачно.

3.7 Восьмеричный формат для целых чисел

Целочисленные значения могут быть записаны в восьмеричной системе счисления. Восьмеричные числовые константы указываются в формате:

```
<0o><восьмеричное число>
```

где `'0'` и `'O'` (или `'o'`) являются литералами, за которыми следует набор байт, использующих 0-7.

Числа состоящие из 1-11 восьмеричных цифр будут интерпретированы как INTEGER, состоящие из 12-21 цифр — как BIGINT, состоящие из 22-43 цифр — как INT128.

```
SELECT  0o12345670 FROM rdb$database;             -- 2739128
SELECT -0o12345670 FROM rdb$database;            -- -2739128
```

Восьмеричные числа в диапазоне 0o0..0o1_7777_777_777 являются положительными INTEGER числами.

Восьмеричные числа в диапазоне 0o20_000_000_000..777_777_777_777_777_777_777 являются положительными BIGINT числами.

Восьмеричные числа в диапазоне от 0o1_000_000_000_000_000_000_000 до 0o1_777_777_777_777_777_777_777_777_777_777_777_777_777_777 являются положительными INT128 числами.

Для представления отрицательных значений нужно поставить знак минус перед беззнаковым восьмеричным литералом.

3.8 Двоичный формат для целых чисел

Целочисленные значения могут быть записаны в двоичной системе счисления. Двоичные числовые константы указываются в формате:

```
<0b><двоичное число>
```

где '0' и 'B' (или 'b') являются литералами, за которыми следует набор байт, использующих 0 или 1.

Числа состоящие из 1-31 двоичных цифр будут интерпретированы как INTEGER, состоящие из 32-63 цифр — как BIGINT, состоящие из 64-43 127 — как INT128.

Для представления отрицательных значений нужно поставить знак минус перед беззнаковым двоичным литералом.

```
SELECT  0b11010100 FROM rdb$database;             -- 212
SELECT -0b11010100 FROM rdb$database;             -- -212
```

3.9 FLOAT

Тип данных FLOAT представляет собой 32-битное число с точностью 7 цифр после запятой (для обеспечения надёжности хранения полагайтесь на 6 цифр).

Диапазон хранимых чисел от 1.175×10^{-38} до 3.402×10^{38}.

Тип данных FLOAT хранится в двоичном формате IEEE 754, который включает в себя знак, показатель степени и мантиссу.

Точность типа FLOAT является динамической, что соответствует физическому формату хранения, который составляет 4 байта.

Этот тип данных (как и другие типы данных с плавающей точкой) не рекомендуется использовать для хранения денежных данных. По тем же причинам не рекомендуется использовать столбцы с данными такого типа в качестве ключей и применять к ним ограничения уникальности.

При проверке данных столбцов рекомендуется вместо точного равенства использовать выражения проверки вхождения в диапазон, например BETWEEN.

Для чисел FLOAT все операции выполняются таким же образом, как принято во всех языках программирования. Следует помнить, что тип данных FLOAT имеет 7 значащих цифр. Если, например,

к очень большому числу FLOAT прибавить очень маленькое число (или вычесть из него такое число), то результат не будет отличаться от первого, большего, числа.

В следующем выражении

```
select cast(cast(1.23E+20 as FLOAT) - cast(1.23E-24 as FLOAT) as float) from rdb
$database;
```

где первый операнд является очень большим по величине числом, а второй — очень маленьким, результатом будет это же первое, большее, число 1.23E+20.

Для числовых типов данных могут использоваться агрегатные функции, определенные в SQL — MIN, MAX, SUM, AVG и множество других встроенных в SQL функций. Описание встроенных функций см. в Приложение З.

3.10 DOUBLE PRECISION

Тип данных DOUBLE PRECISION представляет собой 64-битное число с точностью 15 цифр после запятой.

Диапазон хранимых чисел от 2.225×10^{-308} до 1.797×10^{308}.

Тип данных DOUBLE PRECISION хранится в двоичном формате IEEE 754, который включает в себя знак, показатель степени и мантиссу.

Точность типа DOUBLE PRECISION является динамической, что соответствует физическому формату хранения, который составляет 8 байт.

Этот тип данных (как и другие типы данных с плавающей точкой) не рекомендуется использовать для хранения денежных данных. По тем же причинам не рекомендуется использовать столбцы с данными такого типа в качестве ключей и применять к ним ограничения уникальности.

При проверке данных столбцов рекомендуется вместо точного равенства использовать выражения проверки вхождения в диапазон, например BETWEEN.

Для чисел DOUBLE PRECISION все операции выполняются таким же образом, как принято во всех языках программирования. Следует помнить, что тип данных DOUBLE PRECISION имеет 15 значащих цифр. Если, например, к очень большому числу DOUBLE PRECISION прибавить очень маленькое число (или вычесть из него такое число), то результат не будет отличаться от первого, большего, числа.

В следующем выражении

```
select cast(1.23E+20 as DOUBLE PRECISION) - cast(1.23E-24 as DOUBLE PRECISION) from
rdb$database;
```

где первый операнд является очень большим по величине числом, а второй — очень маленьким, результатом будет это же первое, большее, число 1.23E+20.

Для числовых типов данных могут использоваться агрегатные функции, определенные в SQL — MIN, MAX, SUM, AVG и множество других встроенных в SQL функций. Описание встроенных функций см. в Приложение З.

3.11 DECFLOAT

DECFLOAT является числовым типом из стандарта SQL:2016, который точно хранит числа с плавающей запятой, в отличие от FLOAT или DOUBLE PRECISION, которые обеспечивают двоичное приближение предполагаемой точности. РЕД База Данных в соответствии со стандартом IEEE 754-1985 (IEEE 754-2008) реализует типы DECIMAL64 и DECIMAL128, что обеспечивает точность DECFLOAT 16 и 34 значащих цифр, и занимает 8 и 16 байт памяти соответственно.

Если точность не указана, то по умолчанию используется точность 34 значащих цифры.

Все промежуточные вычисления осуществляются с использованием 34-значными значениями.

Таблица 3.2 — Диапазон значений

Тип	Максимальная точность	Минимальный показатель степени	Максимальный показатель степени	Наименьшее значение	Наибольшее значение
DECFLOAT(16)	16	-383	+384	1E-398	9.9..9E+384
DECFLOAT(34)	34	-6143	+6144	1E-6176	9.9..9E+6144

Все математические, агрегатные и статистические агрегатные функции поддерживают работу со значениями типа DECFLOAT. Кроме этого созданы 4 функции специально для поддержки типа DECFLOAT: COMPARE_DECFLOAT, NORMALIZE_DECFLOAT, QUANTIZE, TOTALORDER.

3.11.1 Режимы округления

Округление происходит, когда точность результата выражения больше точности, поддерживаемой типом данных. DECFLOAT поддерживает все режимы округления, которые требуются в большинстве приложений. Поддерживаются следующие режимы округления совместимые со стандартом IEEE:

Таблица 3.3 — Режим округления

Режим округления	Описание
CEILING	Округление сверху. Если все отбрасываемые цифры равны нулю или знак числа отрицателен, последняя неотбрасываемая цифра не меняется. В противном случае последняя неотбрасываемая цифра инкрементируется на единицу (округляется в большую сторону).
UP	Округление по направлению от нуля (усечение с приращением). Отбрасываемые значения игнорируются.
HALF_UP	Округление к ближайшему значению. Если результат равноудаленный, выполняется округление в большую сторону. Если отбрасываемые значения больше чем или равны половине (0,5) единицы в следующей левой позиции, последняя неотбрасываемая цифра инкрементируется на единицу (округляется в большую сторону). В противном случае отбрасываемые значения игнорируются. Этот режим выбран по умолчанию.
HALF_EVEN	Округление к ближайшему значению. Если результат равноудаленный, выполняется округление так, чтобы последняя цифра была четной. Если отбрасываемые значения больше половины (0,5) единицы в следующей левой позиции, последняя неотбрасываемая цифра инкрементируется на единицу (округляется в большую сторону). Если они меньше половины, результат не корректируется (то есть отбрасываемые знаки игнорируются). В противном случае, когда отбрасываемые значения точно равны половине, последняя неотбрасываемая цифра не меняется, если она является четной и инкрементируется на единицу (округляется в большую сторону) в противном случае (чтобы получить четную цифру). Этот режим округления называется также банковским округлением и дает ощущение справедливого округления.
HALF_DOWN	Округление к ближайшему значению. Если результат равноудаленный, выполняется округление в меньшую сторону. Если отбрасываемые значения больше чем или равны половине (0,5) единицы в следующей левой позиции, последняя неотбрасываемая цифра декрементируется на единицу (округляется в меньшую сторону). В противном случае отбрасываемые значения игнорируются.

(разрыв таблицы)

(разрыв таблицы)

Режим округления	Описание
DOWN	Округление по направлению к нулю (усечение). Отбрасываемые значения игнорируются.
FLOOR	Округление снизу. Если все отбрасываемые цифры равны нулю или знак положителен, последняя неотбрасываемая цифра не меняется. В противном случае (знак отрицателен) последняя неотбрасываемая цифра инкрементируется на единицу.
REROUND	Округление к большему значению, если округляется 0 или 5, в противном случае округление происходит к меньшему значению.

Режим округления можно изменить для текущей сессии используя оператор SET DECFLOAT ROUND:

```
SET DECFLOAT ROUND <режим>
```

По умолчанию используется режим округления HALF_UP.

Таблица 3.4 — Пример округления в разных режимах

Режим округления	12.341	12.345	12.349	12.355	12.405	-12.345
CEILING	12.35	12.35	12.35	12.36	12.41	-12.34
UP	12.35	12.35	12.35	12.36	12.41	-12.35
HALF_UP	12.34	12.35	12.35	12.36	12.41	-12.35
HALF_EVEN	12.34	12.34	12.35	12.36	12.40	-12.34
HALF_DOWN	12.34	12.34	12.35	12.35	12.40	-12.34
DOWN	12.34	12.34	12.34	12.35	12.40	-12.34
FLOOR	12.34	12.34	12.34	12.35	12.40	-12.35
REROUND	12.34	12.34	12.34	12.36	12.41	-12.34

3.11.2 Семантика сравнения

Замыкающие нули в значениях десятичных чисел с плавающей запятой сохраняются. Например, 1.0 и 1.00 — это два различных представления. Это порождает различные семантики сравнения для типа данных DECFLOAT:

- **Сравнение числовых значений**

  Замыкающие нули игнорируются в сравнениях. Например, 1.0 равно 1.00. По умолчанию такой тип сравнения используется для индексирования, сортировки, разбивки таблицы, оценки предикатов и других функций — короче говоря, везде, где сравнение выполняется неявно или в предикатах.

```
create table stockPrice (stock DECFLOAT(16));
insert into stockPrice values (4.2);
insert into stockPrice values (4.2000);
insert into stockPrice values (4.6125);
insert into stockPrice values (4.20);

select * from stockPrice where stock = 4.2;
-----------------------------------------
4.2, 4.2000, 4.20
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
select * from stockPrice where stock > 4.20;
-----------------------------------------
4.6125

select * from stockPrice order by stock;
-----------------------------------------
4.2, 4.20, 4.2000, 4.6125
```

Порядок, в котором возвращаются арифметически одинаковые значения, имеющие различное количество замыкающих нулей, не определен. Следовательно, `ORDER BY` по столбцу `DECFLOAT` со значениями 1.0 и 1.00 возвращает два значения в произвольном порядке. Аналогично, `DISTINCT` возвращает либо 1.0, либо 1.00.

- **Сравнение TotalOrder**

  Замыкающие нули учитываются при сравнении. Например, $1.0 > 1.00$. Каждое значение `DECFLOAT` имеет порядок в семантике сравнения TotalOrder.

  Согласно семантике TotalOrder, порядок различных значений определяется так, как показано в следующем примере:

```
-nan < -snan < -inf < -0.1 < -0.10 < -0 < 0 < 0.10 < 0.1 < inf < snan < nan
```

  Запросить сравнение TotalOrder в предикатах можно при помощи встроенной функции `TOTALORDER()`.

```
create table stockPrice (stock DECFLOAT(16));
insert into stockPrice values (4.2);
insert into stockPrice values (4.2000);
insert into stockPrice values (4.6125);
insert into stockPrice values (4.20);

select * from stockPrice where TOTALORDER(stock, 4.2000) = 0;
-----------------------------------------
4.2000

select * from stockPrice where TOTALORDER(stock, 4.20) = 1;
-----------------------------------------
4.2, 4.6125
```

3.11.3 Обработка ошибок

В процессе вычисления выражений могут возникнуть различные ситуации, которые могут вызвать исключение или проигнорированы. Установить какие ситуации приведут к возбуждению исключения можно с помощью оператора `SET DECFLOAT TRAPS TO`.

```
SET DECFLOAT TRAPS TO <ситуация>[, <ситуация>...]
<ситуация> ::= Division_by_zero
             | Inexact
             | Invalid_operation
             | Overflow
             | Underflow
```

По умолчанию исключения генерируется для следующих ситуаций: `Division_by_zero`,

`Invalid_operation, Overflow`.

Данный SQL оператор работает вне механизма управления транзакциями, изменения выполненные им вступаю в силу немедленно. Его использование разрешено в том числе и PSQL коде.

3.11.4 Поддержка в клиентских приложениях

Библиотека `fbclient` версии 4.0 имеет нативную поддержку типа `DECFLOAT`, в отличии от старых версий клиентской библиотеки. Для того чтобы старые приложения умели работать с типом `DECFLOAT` можно настроить отображение значений `DECFLOAT` на другие доступные типы данных.

```
SET DECFLOAT BIND <тип для привязки>
<тип для привязки> ::= NATIVE
                     | CHAR[ACTER]
                     | DOUBLE PRECISION
                     | BIGINT[, <точность>]
```

Допустимые типы для привязки:

- `NATIVE` — используется IEEE754 двоичное представление. Идеальная поддержка для дальнейшей обработки, но с плохой точностью
- `CHAR[ACTER]` — ASCII строка. Идеальная точность, но плохая поддержка для дальнейшей обработки.
- `DOUBLE PRECISION` — используется 8 байтное представление с плавающей точкой, тоже самое что и для полей типа `DOUBLE PRECISION`. Идеальная поддержка для дальнейшей обработки, но с плохой точностью.
- `BIGINT` — используется целое с указанным масштабом, тоже самое что поле `NUMERIC(18, <точность>)`. Хорошая поддержка дальнейшей обработки и требуемая точность, но диапазон значений очень ограничен.

Привязка к ASCII строке будет иметь тип `CHAR(23)` для `DECFLOAT(16)` и `CHAR(42)` для `DECFLOAT(34)`.

Строковое представление зависит от значения `DECFLOAT`: если оно является экспоненциальным, а требования к точности позволяют отображать значение без использования научной записи, используется полностью выписанный формат.

Данный SQL оператор работает вне механизма управления транзакциями, изменения выполненные им вступаю в силу немедленно. Его использование разрешено в том числе и PSQL коде.

3.11.5 Литералы констант DECFLOAT

Длина литералов типа `DECFLOAT` ограничена 1024 символами. Для более длинных значений вам придётся использовать научную нотацию. Например значение `0.0<1020 нулей>11` не может быть записано как литерал, вместо него вы можете использовать аналогичную научную нотацию: `1.1E-1022`. Аналогично `10<1022 нулей>0` может быть записано как `1.0E1024`.

3.12 NUMERIC

NUMERIC – тип данных с фиксированной точкой, которое ограничивает хранимое число объявленной точностью (количеством чисел после запятой). Формат объявления данных:

```
NUMERIC(<точность>, <масштаб>)

1 <= <точность> <= 38
0 <= <масштаб> <= 38
```

Точность указывает, по меньшей мере, количество цифр для хранения. Масштаб задаёт количество знаков после разделителя.

Масштаб должен быть меньше или равен точности.

Например, NUMERIC(4, 2) описывает число, состоящее в общей сложности из четырёх цифр, включая 2 цифры после запятой; итого 2 цифры до запятой, 2 после. При записи в столбец с этим типом данных значений 3,1415 в столбце NUMERIC(4, 2) будет сохранено значение 3,14.

При сохранении в базу данных число типа NUMERIC умножается на $10^{<>}$, превращаясь в целое. При чтении данных происходит обратное преобразование числа.

Способ физического хранения данных в СУБД зависит от нескольких факторов: декларируемой точности, диалекта базы данных.

Таблица 3.5 — Способ физического хранения чисел NUMERIC

Точность	Диалект 1	Диалект 3
1 - 4	SMALLINT	SMALLINT
5 - 9	INTEGER	INTEGER
10 - 18	DOUBLE PRECISION	BIGINT
19 - 38	INT128	INT128

Всегда надо помнить, что формат хранения данных зависит от точности. Например, вы задали тип столбца NUMERIC(2,2), предполагая, что диапазон значений в нем будет -0.99...0.99. Однако в действительности диапазон значений в столбце будет -327.68..327.67, что объясняется хранением типа данных NUMERIC(2,2) в формате SMALLINT. Фактически типы данных NUMERIC(4, 2), NUMERIC(3,2) и NUMERIC(2,2) являются одинаковыми. Т.е. для реального хранения данных в столбце с типом данных NUMERIC(2,2) в диапазоне -0.99...0.99 для него надо создавать ограничение.

Операции сложения и вычитания для типа данных NUMERIC выполняются обычным образом. Следует только быть особенно внимательными при выполнении операций умножения и деления. В этих операциях результат будет иметь количество дробных знаков, равное сумме дробных знаков обоих операндов. То есть результатом операции деления двух чисел:

```
1.00 / 3
```

будет 0.33. Тот же результат можно получить, если записать операцию деления в следующем виде:

```
1.0 / 3.0
```

Для получения числа с нужной точностью можно также выполнить явное преобразование с использованием функции CAST одного или обоих операндов. Например:

```
CAST(1 AS NUMERIC(3,2)) / 3
```

Результатом будет то же число 0.33.

Для числовых типов данных могут использоваться агрегатные функции, определенные в SQL — `MIN, MAX, SUM, AVG` и множество других встроенных в SQL функций. Описание встроенных функций см. в Приложение 3.

3.13 DECIMAL

`DECIMAL` – тип данных с фиксированной точкой, которое ограничивает хранимое число объявленной точностью (количеством чисел после запятой). Формат объявления данных:

```
DECIMAL(<точность>, <масштаб>)

1 <= <точность> <= 38
0 <= <масштаб> <= 38
```

Точность указывает, по меньшей мере, количество цифр для хранения. Масштаб задаёт количество знаков после разделителя.

Масштаб должен быть меньше или равен точности.

Например, `DECIMAL(4, 2)` описывает число, состоящее в общей сложности из четырёх цифр, включая 2 цифры после запятой; итого 2 цифры до запятой, 2 после. При записи в столбец с этим типом данных значений `3,1415` в столбце `DECIMAL(4, 2)` будет сохранено значение `3,14`.

При сохранении в базу данных число типа `DECIMAL` умножается на $10^{<>}$, превращаясь в целое. При чтении данных происходит обратное преобразование числа.

Способ физического хранения данных в СУБД зависит от нескольких факторов: декларируемой точности, диалекта базы данных.

Таблица 3.6 — Способ физического хранения чисел `DECIMAL`

Точность	Диалект 1	Диалект 3
1 - 4	INTEGER	INTEGER
5 - 9	INTEGER	INTEGER
10 - 18	DOUBLE PRECISION	BIGINT
19 - 38	INT128	INT128

Операции сложения и вычитания для типа данных `DECIMAL` выполняются обычным образом. Следует только быть особенно внимательными при выполнении операций умножения и деления. В этих операциях результат будет иметь количество дробных знаков, равное сумме дробных знаков обоих операндов. То есть результатом операции деления двух чисел

```
1.00 / 3
```

будет 0.33. Тот же результат можно получить, если записать операцию деления в следующем виде:

```
1.0 / 3.0
```

Для получения числа с нужной точностью можно также выполнить явное преобразование с использованием функции `CAST` одного или обоих операндов. Например:

```
CAST(1 AS DECIMAL(3,2)) / 3
```

Результатом будет то же число 0.33.

Для числовых типов данных могут использоваться агрегатные функции, определенные в SQL — MIN, MAX, SUM, AVG и множество других встроенных в SQL функций. Описание встроенных функций см. в Приложение З.

3.14 DATE

В 3-м диалекте тип данных DATE хранит только одну дату без времени. В 1-м диалекте тип DATE эквивалентен типу TIMESTAMP и хранит дату вместе со временем.

Допустимый диапазон хранения от 1 января 1 г. н.э. до 31 декабря 9999 года.

В случае необходимости сохранять в 1 диалекте только значения даты, без времени, при записи в таблицу добавляйте время к значению даты в виде литерала 00:00:00.0000.

В диалекте 1 тип DATE может быть объявлен как TIMESTAMP. Такое объявление является рекомендуемым для новых баз данных в 1-м диалекте.

С типом данных даты возможны арифметические операции сложения и вычитания:

Таблица 3.7 — Арифметические операции с типом данных даты

Операнд 1	Оператор	Операнд 2	Результат
DATE	+	TIME	TIMESTAMP
DATE	+	n	DATE, увеличенная на n целых дней (дробная часть игнорируется)
DATE	-	DATE	Количество дней в интервале DECIMAL (9,0)
DATE	-	n	DATE, уменьшенная на n целых дней (дробная часть игнорируется)

В SQL РЕД Базы Данных существует три предварительно определенных литерала (TODAY, TOMORROW, YESTERDAY) и контекстная переменная CURRENT_DATE типа DATE. Подробнее о них можно найти в Приложение И.

Для работы с типом данных DATE могут использоваться встроенные функции:

- DATEADD позволяет добавить заданное число лет, месяцев, недель, часов, минут, секунд, миллисекунд к заданному значению даты.
- DATEDIFF возвращает количество лет, месяцев, недель, дней, часов, минут, секунд или миллисекунд между двумя значениями даты.
- EXTRACT извлекает составляющие даты из типа данных DATE.
- FIRST_DAY возвращает первый день года, месяца или недели для заданной даты.
- LAST_DAY возвращает последний день года, месяца или недели для заданной даты.

Подробное описание функций см. Приложение З.

3.15 TIME

Этот тип данных доступен только в 3-м диалекте. Позволяет хранить время дня в диапазоне от 00:00:00.0000 до 23:59:59.9999. По умолчанию тип TIME не содержит информацию о часовом поясе. Для того чтобы тип TIME включал информацию о часовом поясе необходимо использовать его с модификатором WITH TIME ZONE.

```
TIME [{WITH | WITHOUT} TIME ZONE]
```

При преобразовании из/в TIME WITH TIME ZONE учитывайте, что тип TIME WITHOUT TIME ZONE определен для использования в часовом поясе сеанса.

Часовой пояс сеанса как следует из названия может быть разным для каждого соединения с базой данных. Он может быть установлен с помощью DPB isc_dpb_session_time_zone, а если нет, то он будет считан из параметра DefaultTimeZone конфигурации firebird.conf. Если параметр DefaultTimeZone не установлен, то часовой пояс сеанса будет тем же, что используется операционной системой в которой запущен процесс СУБД.

Часовой пояс сеанса может быть изменён с помощью оператора SET TIME ZONE или сброшен в исходное значение с помощью SET TIME ZONE LOCAL.

Часовой пояс может быть задан строкой с регионом часового пояса (например, America/Sao_Paulo), или в виде смещения часов:минут относительно GMT (например, -03:00). Список региональных часовых поясов и их идентификаторов см. в Приложение Л. Правила преобразования региональных часовых поясов в смещение в минутах можно получить с помощью процедуры RDB$TIME_ZONE_UTIL.TRANSITIONS.

TIME WITH TIMEZONE считается равным другому TIME WITH TIMEZONE, если их преобразование в UTC равно, например time '10:00 -02' = time '09:00 -03' , поскольку оба времени эквивалентны time '12:00 GMT'. Это также справедливо в контексте ограничения UNIQUE и для сортировки.

TIME WITH TIME ZONE хранится так же как TIME WITHOUT TIME ZONE плюс 2 байта для идентификации часового пояса или смещения. TIME часть хранится в UTC (и переводится в сохранённый часовой пояс).

С типом данных времени возможны арифметические операции сложения и вычитания:

Таблица 3.8 — Арифметические операции с типом данных времени

Операнд 1	Оператор	Операнд 2	Результат
TIME	+	DATE	TIMESTAMP
TIME	+	n	TIME, увеличенная на n секунд (дробная часть учитывается)
TIME	-	TIME	Количество секунд в интервале DECIMAL (9,4)
TIME	-	n	TIME, уменьшенная на n секунд (дробная часть учитывается)

В SQL РЕД Базы Данных существует контекстная переменная CURRENT_TIME типа TIME WITH TIME ZONE и контекстная переменная LOCALTIME типа TIME WITHOUT TIME ZONE. Подробнее о них можно найти в Приложение И.

Для работы с типом данных TIME могут использоваться встроенные функции:

- AT преобразует время в указанный часовой пояс.
- DATEADD позволяет добавить заданное число часов, минут, секунд, миллисекунд к заданному значению времени.
- DATEDIFF возвращает количество часов, минут, секунд или миллисекунд между двумя значениями времени.
- EXTRACT извлекает составляющие времени из типа данных TIME.

Подробное описание функций см. в Приложение З.

3.16 TIMESTAMP

Этот тип данных хранит временную метку (дату вместе со временем) в диапазоне от `01.01.0001 00:00:00.0000` до `31.12.9999 23:59:59.9999`. По умолчанию тип `TIMESTAMP` не содержит информацию о часовом поясе. Для того чтобы тип `TIMESTAMP` включал информацию о часовом поясе необходимо использовать его с модификатором `WITH TIME ZONE`.

```
TIMESTAMP [{WITH | WITHOUT} TIME ZONE]
```

При преобразовании из/в `TIMESTAMP WITH TIME ZONE` учитывайте что тип `TIMESTAMP WITHOUT TIME ZONE` определен для использовании в часовом поясе сеанса (о часовом поясе сеанса см. раздел 3.15).

`TIMESTAMP WITH TIMEZONE` считается равным другому `TIMESTAMP WITH TIMEZONE`, если их преобразование в UTC равно, например `time 10:00 -02 = time 09:00 -03` , поскольку оба времени эквивалентны `time 12:00 GMT` . Это также справедливо в контексте ограничения `UNIQUE` и для сортировки.

`TIMESTAMP WITH TIME ZONE` хранится так же как `TIMESTAMP WITHOUT TIME ZONE` плюс 2 байта для идентификации часового пояса или смещения. `TIMESTAMP` часть хранится в `UTC` (и переводится в сохранённый часовой пояс).

С типом данных `TIMESTAMP` возможны арифметические операции сложения и вычитания:

Таблица 3.9 — Арифметические операции с типом данных `TIMESTAMP`

Операнд 1	Оператор	Операнд 2	Результат
TIMESTAMP	+	n	TIMESTAMP, где дни увеличены на целую часть числа `n`, плюс дробная часть числа `n` (если указана) как количество секунд в дне (с точностью до десятитысячных долей секунды).
TIMESTAMP	-	TIMESTAMP	Количество дней и части дня в интервале `DECIMAL(18, 9)`
TIMESTAMP	-	n	TIMESTAMP, где дни уменьшены на целую часть числа `n`, плюс дробная часть числа `n` (если указана) как количество секунд в дне (с точностью до десятитысячных долей секунды).

В SQL РЕД Базы Данных существует предварительно определенный литерал `NOW`, контекстная переменная `CURRENT_TIMESTAMP` типа `TIMESTAMP WITH TIME ZONE` и контекстная переменная `LOCALTIMESTAMP` типа `TIMESTAMP WITHOUT TIME ZONE`. Подробнее о них можно найти в Приложение И.

Для работы с типом данных `DATE` могут использоваться встроенные функции:

- `AT` преобразует временную метку в указанный часовой пояс.
- `DATEADD` позволяет добавить заданное число лет, месяцев, недель, часов, минут, секунд, миллисекунд к заданному значению даты/времени.
- `DATEDIFF` возвращает количество лет, месяцев, недель, дней, часов, минут, секунд или миллисекунд между двумя значениями даты/времени.
- `EXTRACT` извлекает составляющие даты и времени из типа данных `TIMESTAMP`.
- `FIRST_DAY` возвращает первый день года, месяца или недели для заданной даты.
- `LAST_DAY` возвращает последний день года, месяца или недели для заданной даты.

Подробное описание функций см. в Приложение З.

3.17 CHAR

CHAR является строковым типом данных фиксированной длины. Если введённое количество символом меньше объявленной длины, то поле дополнится концевыми пробелами. В общем случае символ заполнитель может и не являться пробелом, он зависит от набора символов, так например, для набора символов OCTETS — это ноль.

Полное название типа данных CHARACTER, но при работе нет необходимости использовать полные наименования; инструменты по работе с базой прекрасно понимают и короткие имена символьных типов данных.

```
CHAR [(<длина>)] [CHARACTER SET <набор символов>] [COLLATE <сортировка>]
```

В случае если не указана длина, то считается, что она равна единице.

Данный тип символьных данных можно использовать для хранения в справочниках кодов, длина которых стандартна и определённой «ширины». Примером такого может служить почтовый индекс в России – 6 символов.

3.18 VARCHAR

Является базовым строковым типом для хранения текстов переменной длины, поэтому реальный размер хранимой структуры равен фактическому размеру данных плюс 2 байта, в которых задана длина поля.

Если длина не указана, то считается, что она равна 1024.

Все символы, которые передаются с клиентского приложения в базу данных, считаются как значимые, включая начальные и конечные пробельные символы.

Полное название CHARACTER VARYING. Имеется и сокращённый вариант записи CHAR VARYING.

```
VARCHAR [(<длина>)] [CHARACTER SET <набор символов>] [COLLATE <сортировка>]
```

3.19 NCHAR

Представляет собой символьный тип данных фиксированной длины с предопределённым набором символов ISO8859_1.

```
NCHAR [(<длина>)]
```

Синонимом является написание NATIONAL CHAR.

Аналогичный тип данных доступен для строкового типа переменной длины: NATIONAL CHARACTER VARYING.

3.20 Операции и функции для строковых данных

Для строковых типов данных определена операция конкатенации — соединения двух строк в одну. Для обозначения этой операции применяются два подряд идущих символа вертикальной черты ||. Это простая операция, ее результатом является строка, которая представляет собой соединение двух операндов, двух строк. Операция всегда возвращает тип данных CHAR (а не VARCHAR), независимо от того, какой именно строковый тип данных имеют исходные строки. Это означает, что в результате конкатенации сохраняются конечные пробелы. Размер (количество символов) результирующей строки равен сумме размеров исходных строк конкатенации. Например, можно записать следующую операцию:

```
'Руководство ' || 'по SQL'
```

Результатом будет одна строка: "Руководство по SQL".

Для строковых типов данных применимо множество встроенных функций SQL:

- `ASCII_CHAR` возвращает `ASCII` символ соответствующий номеру, переданному в качестве аргумента.
- `ASCII_VAL` возвращает `ASCII` код символа, переданного в качестве аргумента.
- `BIT_LENGTH` возвращает длину входной строки в битах.
- `CHARACTER_LENGTH` возвращает длину (в символах) строки.
- `HASH` возвращает хэш-значение входной строки.
- `LEFT` возвращает левую часть строки.
- `LOWER` возвращает строку в нижнем регистре.
- `LPAD` добавляет к строке слева указанную подстроку.
- `OCTET_LENGTH` возвращает количество байт занимаемое строкой.
- `OVERLAY` заменяет указанное количество символов на заданное значение.
- `POSITION` отыскивает позицию подстроки в строке.
- `REPLACE` отыскивает в строке заданную подстроку и заменяет ее на другую.
- `REVERSE` переписывает символы строки в обратном порядке.
- `RIGHT` возвращает конечную (правую) часть входной строки.
- `RPAD` добавляет к строке справа указанную подстроку.
- `SUBSTRING` возвращает подстроку входной строки.
- `TRIM` удаляет начальные и /или концевые пробелы (или текст согласно настройкам) из входной строки.
- `UPPER` возвращает входную строку в верхнем регистре.

Подробное описание функций см. Приложение З.

Поскольку для строк определена операция сравнения, к строковым данным могут также применяться и агрегатные функции `MIN` и `MAX`. Сравнение строковых данных осуществляется в соответствии с используемым для столбца набором символов (`CHARACTER SET`) и порядком сортировки (`COLLATION ORDER`).

3.21 Строковые литералы

Строковые литералы могут содержать произвольные символы, допустимые для применяемого набора символов. Весь литерал заключается в апострофы. Апостроф внутри символьного литерала должен повторяться два раза, чтобы отличить его от признака завершения литерала. Максимальная длина строкового литерала составляет 65535 байт.

> Необходимо быть осторожным с длиной строки, если значение должно быть записано в столбец типа `VARCHAR`. Максимальная длина строки для типа `VARCHAR` составляет 32765 байт (32767 для типа `CHAR`). Если значение должно быть записано в столбец типа `BLOB`, то максимальная длина строкового литерала составляет 65535 байт.

Пример строковой константы:

```
'Mrs. Hunt''s husband'
```

Вместо двойного (экранированного) апострофа можно использовать другой символ или пару символов.

Ключевое слово `q` или `Q` предшествующее строке в кавычках сообщает парсеру, что некоторые

левые и правые пары одинаковых символов являются разделителями для встроенного строкового литерала.

```
<альтернативный строковый литерал> ::=
      { q | Q } '<левый разделитель> [ { <символ> }... ] <правый разделитель>'
```

Когда `<левый разделитель>` является одним из символов `'('`, `'{'`, `'['` или `'<'`, то `<правый разделитель>` должен быть использован в паре с соответствующим "партнёром" , а именно `')'`, `'}'`, `']'` или `'>'`. В других случаях `<левый разделитель>` совпадает с `<правый разделитель>`.

Внутри строки, т.е. `<символ>`–элементах, одиночные (не экранированные) кавычки могут быть использованы. Каждая кавычка будет частью результирующей строки.

```
SELECT Q'{abc{def}ghi}' FROM rdb$database;          -- abc{def}ghi
SELECT q'!That's a string!' FROM rdb$database;      -- That's a string
```

3.22 Наборы символов

Полный список доступных наборов символов и нестандартных порядков сортировки доступен в Приложение Д.

Набор символов и порядок сортировки задаются:

- при создании базы данных
- при описании текстового объекта базы данных
- при соединении клиентской программы к базе данных

В случае отсутствия явного указания набора символов при описании текстового объекта базы данных будет использоваться набор символов по умолчанию, заданный при создании базы данных. При отсутствии явного указания набора символов, а также отсутствия набора символов по умолчанию в базе данных, поле получает набор символов `CHARACTER SET NONE`.

В случае различия набора символов клиентского соединения и текстового поля базы данных, при выдаче результата для строковых столбцов происходит автоматическая перекодировка как при передаче данных с клиента на сервер, так и в обратном направлении с сервера на клиента. То есть, совершенно нормальной является ситуация, когда база создана в кодировке `WIN1251`, а в настройках клиента в параметрах соединения стоит `KOI8R` или `UTF8`.

3.22.1 UNICODE

При возникновении необходимости использования восточноевропейских текстов в строковых полях базы данных или для более экзотических алфавитов, рекомендуется работать с набором символов `UTF8`. При этом следует иметь в виду, что на один символ в данном наборе приходится до 4 байт. Следовательно, максимальный размер символов в символьных полях составит 32676/4 (8192) байта на символ. При этом следует обратить внимание, что фактически значение параметра «байт на символ» зависит от диапазона, к которому принадлежит символ: английские буквы занимают 1 байт, русские буквы кодировки `WIN1251` — 2 байта, остальные символы — могут занимать до 4-х байт.

Набор символов `UTF8` поддерживает последнюю версию стандарта `Unicode`, до 4 байт на символ, поэтому для интернациональных баз рекомендуется использовать именно эту реализацию поддержки `Unicode` в Firebird.

Если база данных будет содержать строки только с русским алфавитом, то для неё рекомендуется к использованию кодировка `WIN1251`. При её использовании на один символ расходуется 1 байт, соответственно максимальный размер текстовых полей для данной кодировки будет 32767 символов. Для стандартных операций сортировки при работе с `WIN1251` не требуется задавать порядок сортировки (`COLLATE`).

3.22.2 Набор символов NONE и OCTETS

Набор символов NONE относится к специальным наборам символов. Его можно охарактеризовать тем, что каждый байт является частью строки, но в системе хранится без указаний, к какому фактическому набору символов они относятся. Разбираться с такими данными должно клиентское приложение, на него возлагается ответственность в правильной трактовке символов из таких полей.

Также к специальным наборам символов относится OCTETS. В этом случае данные рассматриваются как байты, которые могут в принципе не интерпретироваться как символы. OCTETS позволяет хранить бинарные данные и/или результаты работы некоторых функций Firebird. Правильное отображение данных пользователю, хранящихся в полях с CHARACTER SET OCTETS, также становится заботой клиентской стороны. При работе с подобными данными следует также помнить, что СУБД не контролирует их содержимое и возможно возникновение исключения при работе кода, когда идёт попытка отображения бинарных данных в желаемой кодировке.

3.22.3 Порядок сортировки

Каждый текстовый набор символов (CHARACTER SET) имеет последовательность сортировки (COLLATE) по умолчанию, задающий порядок сортировки и способы сравнения. Если необходимо нестандартное поведение строк при указанных выше действиях, то в описании строкового столбца может быть указан параметр COLLATE, который его опишет. Помимо описания объявления столбца, выражение COLLATE может быть добавлено в предложениях SELECT в секции WHERE, когда происходят операции сравнения больше — меньше, в секции ORDER BY при сортировке по символьному полю, а также при операциях группировки для указания специальной последовательности сортировки при выводе в предложении GROUP BY.

Регистронезависимый поиск

Для регистронезависимого поиска можно воспользоваться функцией UPPER:

```
WHERE UPPER(name) = UPPER(:flt_name)
```

Для строк с набором символов WIN1251 можно для этих же целей воспользоваться предложением COLLATE PXW_CYRL.

```
WHERE FIRST_NAME COLLATE PXW_CYRL >= :FLT_NAME
```

```
ORDER BY NAME COLLATE PXW_CYRL
```

Порядок сортировки для UTF-8

Ниже приведена таблица возможных последовательностей сортировки для набора символов UTF8.

Таблица 3.10 — Последовательности сортировки для UTF8

COLLATION	Комментарии
UCS_BASIC	Сортировка работает в соответствии с положением символа в таблице (бинарная). Пример: A, B, a, b, a...
UNICODE	Сортировка работает в соответствии с алгоритмом UCA (Unicode Collation Algorithm) (алфавитная). Пример: a, A, a, b, B...
UTF-8	Сортировка происходит без учёта регистра символа.

(разрыв таблицы)

(разрыв таблицы)

COLLATION	Комментарии
UNICODE_CI_AI	Сортировка происходит без учёта регистра символа, в алфавитном порядке.

3.22.4 Индексирование символьных типов

При построении индекса по строковым полям необходимо учитывать ограничение на длину ключа индекса. Максимальная используемая длина ключа индекса равна 1/4 размера страницы, т.е. от 1024 до 4096 байтов. Максимальная длина индексируемой строки на 9 байтов меньше, чем максимальная длина ключа. В таблице приведены данные для максимальной длины индексируемой строки (в символах) в зависимости от размера страницы и набора символов, её можно вычислить по следующей формуле:

```
max_char_length = FLOOR((page_size / 4 - 9) / N)
```

где N — число байтов на представление символа.

Размер страницы	Максимальная длина индексируемой строки для набора символов, байт/символ				
	1	2	3	4	6
4096	1015	507	338	253	169
8192	2039	1019	679	509	339
16384	4087	2043	1362	1021	682
32768	9183	4087	2721	2039	1356

В кодировках, нечувствительных к регистру ("_CI"), один символ в *индексе* будет занимать не 4, а 6 байт

Последовательность сортировки (COLLATE) тоже может повлиять на максимальную длину индексируемой строки.

3.23 BOOLEAN

РЕД База Данных предоставляет стандартный SQL тип BOOLEAN. Значений у этого типа может быть несколько: TRUE (истина), FALSE (ложь) и третье состояние UNKNOWN (неизвестно) представляется SQL значением NULL. Спецификация не делает различия между значением NULL этого типа и значением истинности UNKNOWN, которое является результатом SQL предиката, поискового условия или выражения логического типа. Эти значения взаимозаменяемы и обозначают одно и то же.

Значения типа BOOLEAN могут быть проверены в неявных значениях истинности. Например, field1 OR field2 или NOT field1 являются допустимыми выражениями.

Предикаты могут использовать оператор IS [NOT] для проверки соответствия. Например, field1 IS FALSE или field1 IS NOT TRUE.

Операторы эквивалентности (=, !=, <> и др.) допустимы во всех сравнениях. Значение TRUE больше чем FALSE.

Тип данных BOOLEAN не преобразуется неявно ни к одному типу, но возможно явное преобразование к строке с помощью функции CAST.

Пример

Пример использования типа BOOLEAN:

```
create table CHECKBOOL (ID integer, BOOLVAL boolean);
insert into CHECKBOOL values (1, 1 != 4);
insert into CHECKBOOL values (2, FALSE);
insert into CHECKBOOL values (3, NULL - 1);

SELECT * FROM CHECKBOOL;

ID BOOLVAL
============ =======
1 <true>
2 <false>
3 <null>

SELECT * FROM CHECKBOOL WHERE BOOLVAL = TRUE;

ID BOOLVAL
============ =======
1 <true>

SELECT * FROM CHECKBOOL WHERE NOT BOOLVAL;

ID BOOLVAL
============ =======
2 <false>

SELECT * FROM CHECKBOOL WHERE BOOLVAL IS UNKNOWN;

ID BOOLVAL
============ =======
3 <null>
```

3.24 BINARY

BINARY является типом данных с фиксированной длиной для хранения бинарных данных. Если переданное количество байт меньше объявленной длины, то значение будет дополнено нулями. В случае если не указана длина, то считается, что она равна единице.

```
BINARY [(<длина>)]
```

Этот тип является псевдонимом типа CHAR [(<длина>)] CHARACTER SET OCTETS и обратно совместим с ним.

Данный тип хорошо подходит для хранения уникального идентификатора полученного с помощью функции GEN_UUID.

3.25 VARBINARY

VARBINARY является типом для хранения бинарных данных переменной длины. Реальный размер хранимой структуры равен фактическому размеру данных плюс 2 байта, в которых задана длина поля.

Полное название BINARY VARYING.

```
VARBINARY (<длина>) | BINARY VARYING (<длина>)
```

Этот тип является псевдонимом типа `VARCHAR (<длина>) CHARACTER SET OCTETS` и обратно совместим с ним.

3.26 BLOB

Тип данных `BLOB` называется большим двоичным объектом (Binary Large OBject). Этот тип данных позволяет хранить любые очень большие по объему данные — форматированные тексты, графику, звуки, видео.

Листинг 3.1. Синтаксис объявления типа `BLOB`

```
BLOB [SUB_TYPE <имя подтипа>]
[SEGMENT SIZE <размер сегмента>]
[CHARACTER SET <набор символов>]
```

Также можно использовать сокращенный синтаксис:

Листинг 3.2. Сокращенный синтаксис объявления типа `BLOB`

```
BLOB (<размер сегмента>) |
BLOB (<размер сегмента>, <имя подтипа>) |
BLOB (, <имя подтипа>)
```

Тип данных `BLOB` характеризуется с точки зрения хранения в базе данных, в первую очередь, размером сегмента. Максимальный размер сегмента не может превышать 64Кб – 1, то есть числа 65535. Объем данных, которые могут храниться в этом типе данных, зависит от размера страницы базы данных:

- При размере страницы 4096 байтов размер `BLOB` не может превышать 4 ГБ,
- При размере страницы 8192 байтов — 32 ГБ,
- При размере страницы 16384 байтов — 256 ГБ.

При объявлении столбца или домена типа `BLOB` можно указать его подтип (предложение `SUB_TYPE`), а также размер сегмента, используемый при хранении данных (предложение `SEGMENT SIZE`). Значение подтипа может быть целым числом в диапазоне от –32768 до +32767.

Подтипы (положительные числа или 0) могут использоваться в случае, когда в базе данных описаны стандартные `BLOB`-фильтры. Фильтры — это программы, которые выполняют преобразования между данными `BLOB` разных подтипов на серверной и клиентской стороне. Такие преобразования связаны, как правило, с упаковкой и, соответственно, распаковкой данных.

В РЕД Базе Данных существуют семь заранее предопределенных подтипов. Они представлены в таблице 3.12. Их не следует использовать для каких-то своих внутренних целей.

Таблица 3.12 — Предопределенные подтипы типа BLOB

Подтип BLOB	Имя подтипа	Назначение
0	BINARY	Если подтип не указан, то данные считаются нетипизированными и значение подтипа принимается равным 0. Этот подтип указывает, что данные имеют форму бинарного файла или потока (изображение, звук, видео, файлы текстового процессора, PDF и т.д.).
1	TEXT	Это специализированный подтип, который используется для хранения текстовых данных большого объёма. Для текстового подтипа BLOB может быть указан набор символов и порядок сортировки COLLATE, аналогично символьному полю.
2	BLR	Данные двоичного представления языка (BLR - binary language representation)
3	ACL	Список управления доступом
4	RANGES	Резервируется для будущих использований
5	SUMMARY	Закодированные описания для метаданных таблиц
6	FORMAT	Форматированные данные
7	TRANSACTION_DESCRIPTION	Описание транзакций ко многим базам данных. Эти транзакции завершаются в неопределенном порядке
8	EXTERNAL_FILE_DESCRIPTION	Описание внешних файлов

Для пользовательских подтипов рекомендуется выбирать только отрицательные числа, поскольку положительные могут использоваться системой, в том числе и при дальнейших расширениях. На клиентских программах лежит ответственность за то, что в поля BLOB заданного подтипа записываются данные соответствующего вида.

3.26.1 Хранение BLOB

Поля BLOB не хранятся непосредственно в самой записи вместе с другими данными строки. Запись содержит только ссылку (идентификатор, указатель) на страницу базы данных, где располагаются данные BLOB. Сами данные помещаются в сегменты. Размер сегмента задает в байтах размер полей в базе данных, которые будут использованы для хранения данных типа BLOB. По умолчанию принимается 80. Максимально возможное значение 65535. За одно обращение к базе данных система всегда считывает один сегмент. Если в поле BLOB хранятся данные, занимающие менее 32765 байтов, то хранение и работа с этим полем осуществляется так же, как и с полем, имеющим тип данных VARCHAR.

По умолчанию, для каждого BLOB создаётся обычная запись, хранящаяся на какой-то выделенной для этого странице данных (data page). Если весь BLOB на эту страницу поместится, его называют BLOB уровня 0. Номер этой специальной записи хранится в записи таблицы и занимает 8 байт.

Если BLOB не помещается на одну страницу данных (data page), то его содержимое размещается на отдельных страницах, целиком выделенных для него (blob page), а в записи о BLOB помещают номера этих страниц. Это BLOB уровня 1.

Если массив номеров страниц с данными BLOB не помещается на страницу данных (data page), то его (массив) размещают на отдельных страницах (blob page), а в запись о BLOB помещают уже номера этих страниц. Это BLOB уровня 2.

Уровни выше 2 не поддерживаются.

3.27 Массивы

Поддержка массивов в СУБД Firebird является расширением традиционной реляционной модели. Поддержка в СУБД такого инструмента позволяет проще решать некоторые задачи по обработке однотипных данных. Массивы в Firebird реализованы на базе полей типа `BLOB`. Массивы могут быть одномерными и многомерными.

```
CREATE TABLE SAMPLE_ARR (
       ID INTEGER NOT NULL PRIMARY KEY,
       ARR_INT INTEGER [4]);
```

Так будет создана таблица с полем типа массива из четырёх целых. Индексы данного массива от 1 до 4. Для определения верхней и нижней границы значений индекса следует воспользоваться следующим синтаксисом:

```
[ <нижняя>: <верхняя>]
```

Добавление новой размерности в синтаксисе идёт через запятую. Пример создания таблицы с массивом размерности два, в котором нижняя граница значений начинается с нуля:

```
CREATE TABLE SAMPLE_ARR2 (
       ID INTEGER NOT NULL PRIMARY KEY,
       ARR_INT INTEGER [0:3, 0:3]);
```

СУБД не предоставляет большого набора инструментов для работы с содержимым массивов. База данных `employee.fdb`, которая находится в дистрибутиве РЕД Базы Данных, содержит пример хранимой процедуры, показывающей возможности работы с массивами. Ниже приведён её текст:

```
ALTER PROCEDURE SHOW_LANGS (
     CODE VARCHAR(5),
     GRADE SMALLINT,
     CTY VARCHAR(15))
RETURNS (
     LANGUAGES VARCHAR(15))
AS
     DECLARE VARIABLE I INTEGER;
BEGIN
     i = 1;
     WHILE (i <= 5) DO
     BEGIN
           SELECT language_req[:i] FROM job
           WHERE ((job_code = :code) AND (job_grade = :grade) AND (job_country = :cty)
                   AND (language_req IS NOT NULL))
           INTO :LANGUAGES;
           IF (LANGUAGES = ' ') THEN  /* Prints 'NULL' instead of blanks */
              LANGUAGES = 'NULL';
           I = I +1;
           SUSPEND;
     END
END
```

Если приведённых выше возможностей достаточно для ваших задач, то вы вполне можете применять массивы для своих проектов. В настоящее время совершенствования механизмов обработки массивов средствами СУБД не производится.

3.28 SQL_NULL

Тип данных SQL_NULL содержит не данные, а только состояние: NULL или NOT NULL. Также этот тип данных не может быть использован при объявлении полей таблицы, переменных или PSQL параметров. Этот тип данных добавлен для улучшения поддержки нетипизированных параметров в предикате IS NULL. Такая проблема возникает при использовании «отключаемых фильтров» при написании запросов следующего типа:

```
WHERE col = :param OR :param IS NULL
```

Приведем пример. Разработчики приложений хотят поддерживать запросы с дополнительными фильтрами, такими, как эти:

```
SELECT
     AU.MAKE, AU.MODEL, AU.WEIGHT, AU.PRICE, AU.IN_STOCK
FROM AUTOMOBILES AU
WHERE (AU.MAKE = :MAKE OR :MAKE IS NULL)
    AND (AU.MODEL = :MODEL OR :MODEL IS NULL)
    AND (AU.PRICE <= :MAXPRICE OR :MAXPRICE IS NULL)
```

Идея состоит в том, что конечный пользователь может дополнительно ввести варианты для параметров :MAKE, :MODEL и :MAXPRICE. Там, где сделан выбор, должен быть применен соответствующий фильтр. Везде, где значение параметра не установлено (NULL), никакой фильтрации по этому атрибуту не должно быть. Если все параметры не установлены, то должны быть показана вся таблица AUTOMOBILES.

Именованные параметры, такие как :MAKE, :MODEL и :MAXPRICE, существует только на уровне приложений. Прежде чем запрос передается серверу для подготовки, он должен быть преобразован в такую форму:

```
SELECT
   AU.MAKE, AU.MODEL, AU.WEIGHT, AU.PRICE, AU.IN_STOCK
FROM AUTOMOBILES AU
WHERE  (AU.MAKE = ? OR ? IS NULL)
   AND (AU.MODEL = ? OR ? IS NULL)
   AND (AU.PRICE <= ? OR ? IS NULL)
```

Вместо трех именованных параметров, каждый из которых используется два раза, мы теперь имеем шесть позиционных параметров. РЕД База Данных не может определить тип данных параметра ? IS NULL. Эта проблема может быть решена путем приведения типов, например, WHERE (AU.MAKE = ? OR CAST(? AS TYPE OF COLUMN AU.MAKE) IS NULL).... Но это довольно громоздко. И еще одна проблема: там, где параметр для фильтра не NULL, его значение будет передано серверу два раза. А это небольшие, но потери производительности.

Для решения этих проблем и был введён тип данных SQL_NULL.

3.29 Явное преобразование типов данных. Функция CAST

Хотя в некоторых случаях сервер РЕД Базы Данных осуществляет неявное преобразование данных в подходящий тип, тем не менее, во избежание неверных результатов или других возможных ошибок всегда следует осуществлять явное преобразование типов данных при выполнении операций, включая операции сравнения, над данными различных типов.

В тех случаях, когда требуется выполнить явное преобразование одного типа в другой, исполь-

зуют функцию CAST.

Функция CAST позволяет преобразовывать данные из одного типа данных в другой, допустимый для исходного значения. Синтаксис функции:

Листинг 3.3. Синтаксис функции преобразования типов данных CAST

```
CAST ( {<значение> | NULL} AS <тип данных> [CHARACTER SET <набор символов>] )

<тип данных> ::= {
            <тип данных SQL>
          | [TYPE OF] <имя домена>
          | TYPE OF COLUMN <имя таблицы/представления>.<имя столбца> }
```

При преобразовании к домену ([TYPE OF] <имя домена>) учитываются объявленные для него ограничения, например, NOT NULL или описанные в CHECK и если <значение> не пройдёт проверку, то преобразование не удастся. В случае если дополнительно указывается TYPE OF (преобразование к базовому типу), при преобразовании игнорируются любые ограничения домена. При использовании TYPE OF с типом (VAR)CHAR набор символов и сортировка сохраняются.

При преобразовании к типу столбца (TYPE OF COLUMN) допускается использовать указание столбца таблицы или представления. Используется только сам тип столбца; в случае строковых типов это также включает набор символов, но не сортировку. Ограничения и значения по умолчанию исходного столбца не применяются.

При преобразовании любого типа в строковый тип данных CHAR или VARCHAR можно также указать и набор символов, в который переводится строка.

Преобразование константы NULL в любой тип данных всегда дает NULL.

Таблица 3.13 — Допустимые преобразования для функции CAST

Из типа	В тип
Числовые типы (SMALLINT, INTEGER, BIGINT, FLOAT, DOUBLE PRECISION, DECFLOAT, NUMERIC, DECIMAL)	Числовые типы, [VAR]CHAR, BLOB
[VAR]CHAR, BLOB	[VAR]CHAR, BLOB, BOOLEAN, Числовые типы, DATE, TIME, TIMESTAMP
DATE, TIME	[VAR]CHAR, BLOB, TIMESTAMP
TIMESTAMP	[VAR]CHAR, BLOB, TIME, DATE
BOOLEAN	[VAR]CHAR, BLOB

Для преобразования строковых типов данных в тип BOOLEAN необходимо чтобы строковый аргумент был одним из предопределённых литералов логического типа (true или false).

При преобразовании типов следует помнить о возможной частичной потери данных, например, при преобразовании типа данных TIMESTAMP в DATE.

3.29.1 Преобразование в типы данных даты и времени

Для преобразования строковых типов данных в типы DATE, TIME или TIMESTAMP с помощью функции CAST необходимо, чтобы строковый аргумент был либо одним из предопределённых литералов даты и времени, либо строковое представление даты в одном из разрешённых форматов.

```
<литерал даты> ::=
      [ГГГГ<р>]ММ<р>ДД |
      ММ<р>ДД[<р>ГГГГ] |
      ДД<р>ММ[<р>ГГГГ] |
      ММ<р>ДД[<р>ГГ] |
      ДД<р>ММ[<р>ГГ]

<литерал времени> := ЧЧ[:мм[:СС[.ННННН]]]

<литерал даты-времени> ::= <литерал даты> <литерал времени>

<часовой пояс> ::=
      <региональный часовой пояс> |
      [+/-] <разница часов с GMT> : <разница минут с GMT>

<р> ::= <пробел> | . | : | , | - | /
```

Таблица 3.14 — Описание формата даты и времени

Аргумент	Описание
ГГГГ	Год из четырёх цифр
ГГ	Последние две цифры года (00-99)
ММ	Номер месяца в 1 или 2 цифры (1-12 или 01-12). В качестве месяца допустимо также указывать трёхбуквенное сокращение (`jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec`) или полное наименование месяца на английском языке, регистр не имеет значение
ДД	День месяца в 1 или 2 цифры (1-31 или 01-31)
ЧЧ	Час в 1 или 2 цифры (0-23 или 00-23)
мм	Минуты 1 или 2 цифры (0-59 или 00-59)
СС	Секунды в 1 или 2 цифры (0-59 или 00-59)
ННННН	Десятитысячные доли секунды от 1 до 4 цифр (0-9999)

Правила:

- В формате ГГГГ<р>ММ<р>ДД, год обязательно должен содержать 4 цифры;
- Для дат в формате с завершающим годом, если в качестве разделителя дат используется точка ".", то дата интерпретируется в форме День-Месяц-Год, для остальных разделителей она интерпретируется в форме Месяц-День-Год;
- Если год не указан, то в качестве года берётся текущий год;
- Если указаны только две цифры года, то для получения столетия Firebird использует алгоритм скользящего окна. Задача заключается в интерпретации двухсимвольного значения года как ближайшего к текущему году в интервале предшествующих и последующих 50 лет;
- Если не указан один из элементов времени, то оно принимается равным 0.

Настоятельно рекомендуем в литералах дат использовать только формы с полным указанием года в виде 4 цифр во избежание путаницы.

Примеры преобразований в типы данных даты-времени:

```
CAST('04.12.2014' AS DATE)      -- ДД.ММ.ГГГГ
CAST('12-04-2014' AS DATE)      -- ММ-ДД-ГГГГ
CAST('12/04/2014' AS DATE)      -- ММ/ДД/ГГГГ
CAST('04.12.14' AS DATE)        -- ДД.ММ.ГГ
CAST('04.12' AS DATE)           -- ДД.ММ  в качестве года берётся текущий
CAST('12/4' AS DATE)            -- ММ/ДД  в качестве года берётся текущий
CAST('2014/12/04' AS DATE)      -- ГГГГ/ММ/ДД
CAST('2014.12.04' AS DATE)      -- ГГГГ.ММ.ДД
CAST('2014-12-04' AS DATE)      -- ГГГГ-ММ-ДД
CAST('11:37' AS TIME)           -- ЧЧ:мм
CAST('11:37:12' AS TIME)        -- ЧЧ:мм:сс
CAST('11:31:12.1234' AS TIME)   -- ЧЧ:мм:сс.нннн
CAST('11:31:12.1234 +03:30' AS TIME WITH TIME ZONE)          -- ЧЧ:мм:сс.нннн +чч:мм
CAST('11:31:12.1234 Europe/Moscow' AS TIME WITH TIME ZONE)   -- ЧЧ:мм:сс.нннн тз
CAST('11:31 Europe/Moscow' AS TIME WITH TIME ZONE)           -- ЧЧ:мм тз
CAST('04.12.2014 11:37' AS TIMESTAMP)            -- ДД.ММ.ГГГГ ЧЧ:мм
CAST('12/04/2014 11:37:12' AS TIMESTAMP)         -- ММ/ДД/ГГГГ ЧЧ:мм:сс
CAST('04.12.2014 11:31:12.1234' AS TIMESTAMP)    -- ДД.ММ.ГГГГ ЧЧ:мм:сс.нннн
CAST('2014-12-04 11:31:12.1234 +03:00' AS TIMESTAMP WITH TIME ZONE)
-- ГГГГ-ММ-ДД ЧЧ:мм:сс.нннн +чч:мм
CAST('04.12.2014 11:31:12.1234 Europe/Moscow' AS TIMESTAMP WITH TIME ZONE)
-- ДД.ММ.ГГГГ ЧЧ:мм:сс.нннн тз
```

Помимо функции `CAST` можно использовать декларации типа константы (чтобы объявить именно дату, а не строку):

```
{ TIMESTAMP | DATE | TIME } '<литерал даты-времени>'
```

Разница константы и преобразования типа в том, что константа определяется на уровне парсинга запроса, а преобразование выполняется в рантайме. Разницу очень хорошо видно в статистике выполнения:

- `a = '01.01.2000'`
- `a = cast('01.01.2000' as date)`
- `a = date '01.01.2000'`

При большом числе проверяемых записей последний вариант сильно быстрее.

```
UPDATE SALES
SET ORDER_STATUS = 'shipped'
WHERE ORDER_DATE > DATE '1-Jan-1993';

INSERT INTO EMPLOYEE (EMP_NO, DEPT_NO, HIRE_DATE)
VALUES (973, 8804, DATE '05.12.2023' + 2 + TIME '16:00');

INSERT INTO EMPLOYEE (EMP_NO, DEPT_NO, HIRE_DATE)
VALUES (34, 884, TIMESTAMP '01-22-2100');
```

Обратите внимание, что эти сокращённые выражения вычисляются сразу же во время синтаксического анализа, то есть, как будто оператор уже подготовлен к выполнению. Таким образом, даже если запрос выполняется несколько раз, значение, например, для `TIMESTAMP 'NOW'` не изменится, независимо от того, сколько времени проходит. Если вам нужно получать нарастающее значение времени (т.е. оно должно быть оценено при каждом вызове), используйте полный синтаксис `CAST`.

```
NEW.CHANGE_DATE = CAST(NOW AS TIMESTAMP);
```

Таблица 3.15 — Литералы с предопределёнными значениями даты и времени

Литерал	Значение	Тип данных для диалекта 1	Тип данных для диалекта 3
'NOW'	Текущая дата и время	DATE	TIMESTAMP
'TODAY'	Текущая дата	DATE (с нулевым временем)	DATE (только дата)
'TOMORROW'	Завтрашняя дата	DATE (с нулевым временем)	DATE (только дата)
'YESTERDAY'	Вчерашняя дата	DATE (с нулевым временем)	DATE (только дата)

3.29.2 Преобразование из типов данных даты и времени в строку

Синтаксис для преобразования из типов данных даты и времени в строковые типы:

Листинг 3.4. Синтаксис функции преобразования из типов данных даты и времени в строку

```
CAST (<значение> AS <тип данных> FORMAT <шаблон>)

<значение> ::=
    DATE
  | TIME [{WITH | WITHOUT} TIME ZONE]
  | TIMESTAMP [{WITH | WITHOUT} TIME ZONE]

<тип данных> ::=
    VARCHAR
  | CHAR

<шаблон> ::= ["строковое значение"] <формат>
```

Таблица 3.16 — Описание формата даты и времени

Формат	Описание
YEAR	Год (1 - 9999)
YYYY YYY YY Y	Последние 4 цифры года (0001 - 9999) Последние 3 цифры года (000 - 999) Последние 2 цифры года (00 - 99) Последняя цифра года (0 - 9)
Q	Квартал года (1 - 4)
MM	Месяц (01 - 12)
MON	Сокращённое название месяца (Apr)
MONTH	Название месяца (APRIL)
RM	Римское представление месяца (I - XII)

(разрыв таблицы)

(разрыв таблицы)

Формат	Описание
WW	Неделя года (01 - 53)
W	Неделя месяца (1 - 5)
D	День недели (1 - 7)
DAY	Название дня недели (MONDAY)
DD	День месяца (01 - 31)
DDD	День года (001 - 366)
DY	Сокращённое название дня недели (Mon)
J	Юлианский день (кол-во дней с 1 января 4712г. до н.э.)
HH HH12	Часы (01 - 12) без интервала (для указания интервала используйте A.M., P.M.)
HH24	Часы (00 - 23)
MI	Минуты (00 - 59)
SS	Секунды (00 - 59)
SSSSS	Секунды после полуночи (0 - 86399)
FF1 FF2 FF3 FF4 FF5 FF6 FF7 FF8 FF9	Дробные секунды с указанной точностью
A.M. P.M.	Интервал для времени в формате HH12 (не имеет значения, какой из них используется, период будет вставлен в зависимости от времени)
TZH	Часовой пояс в часах (-14 - 14)
TZM	Часовой пояс в минутах (00 - 59)
TZR	Название часового пояса

В качестве разделителей можно использовать следующие символы: '.', '/', ',', ';', ':', '<пробел>', '-'.

Форматы можно указывать без разделителей:

```
SELECT CAST(CURRENT_TIMESTAMP AS VARCHAR(50)
                   FORMAT 'YEARMMDD HH24MISS') FROM RDB$DATABASE;   -- 20230719 161757
```

Однако, при указании форматов без разделителей нужно быть внимательнее, например, 'DDDDD' будет интерпретирован как 'DDD' и 'DD'.

Формат не чувствителен к регистру.

Пример преобразования времени в строку:

```
SELECT CAST(CURRENT_TIMESTAMP AS VARCHAR(20)
                   FORMAT 'HH24:MI:SS') FROM RDB$DATABASE;   -- 15:13:50
```

В формате можно указывать строковое значение:

```
SELECT CAST(CURRENT_TIMESTAMP AS VARCHAR(20)
            FORMAT '"Today is" DAY') FROM RDB$DATABASE;   -- Today is TUESDAY
```

Экранируйте символ двойных кавычек (\"), чтобы добавить его в строку вывода. Для печати \ используйте \\.

Пример преобразования времени в Юлианский день:

```
SELECT CAST(CURRENT_TIMESTAMP AS VARCHAR(45)
             FORMAT 'DD.MM.YEAR HH24:MI:SS "is" J "Julian day"') FROM RDB$DATABASE;
-- 14.6.2023 15:41:29 is 2460110 Julian day
```

3.29.3 Преобразование из строки в типы данных даты и времени

Синтаксис для преобразования из строки в типы данных даты и времени:

Листинг 3.5. Синтаксис функции преобразования из строки в типы данных даты и времени

```
CAST (<значение> AS <тип данных> FORMAT <шаблон>)

<значение> ::=
    VARCHAR
  | CHAR

<тип данных> ::=
      DATE
    | TIME [{WITH | WITHOUT} TIME ZONE]
    | TIMESTAMP [{WITH | WITHOUT} TIME ZONE]

<шаблон> ::= ["строковое значение"] <формат>
```

Таблица 3.17 — Описание формата даты и времени

Формат	Описание
YEAR	Год (1 - 9999)
YYYY YYY YY Y	Последние 4 цифры года (0001 - 9999) Последние 3 цифры года (000 - 999) Последние 2 цифры года (00 - 99) Последняя цифра года (0 - 9)

(разрыв таблицы)

(разрыв таблицы)

Формат	Описание
RR	Округление года. При использовании RR значение округляется по следующему правилу: Указанное значение от 00 до 49: • Если последние две цифры текущего года от 00 до 49, то возвращаемый год будет иметь те же первые две цифры, что и текущий год; • Если последние две цифры текущего года от 50 до 99, то первые 2 цифры возвращаемого года будут на 1 больше первых 2 цифр текущего года. Указанное значение от 50 до 99: • Если последние две цифры текущего года от 00 до 49, то первые 2 цифры возвращаемого года будут на 1 меньше первых 2 цифр текущего года. • Если последние две цифры текущего года от 50 до 99, то возвращаемый год будет иметь те же первые две цифры, что и текущий год.
RRRR	Последние 4 цифры года (0001 - 9999)
MM	Месяц (1 - 12)
MON	Сокращённое название месяца (Apr)
MONTH	Название месяца (APRIL)
RM	Римское представление месяца (I - XII)
DD	День месяца (1 - 31)
J	Юлианский день (кол-во дней с 1 января 4712г. до н.э.)
HH HH12	Часы (1 - 12) без интервала (для указания интервала используйте A.M., P.M.)
HH24	Часы (0 - 23)
MI	Минуты (0 - 59)
SS	Секунды (0 - 59)
SSSSS	Секунды после полуночи (0 - 86399)
FF1 FF2 FF3 FF4	Дробные секунды с указанной точностью
A.M. P.M.	Интервал для времени в формате HH12 (не имеет значения, какой из них используется, период будет взят из входной строки)
TZH	Часовой пояс в часах (-14 - 14)
TZM	Часовой пояс в минутах (0 - 59)
TZR	Название часового пояса

В качестве разделителей можно использовать следующие символы: '.', '/', ',', ';', ':', '<пробел>', '-'.

Год, месяц и день будут взяты из текущей даты, если они не используются в шаблоне.

Пример преобразования:

```
SELECT CAST('2000.12.08 12:35:30.5000' AS TIMESTAMP
            FORMAT 'YEAR-MM-DD HH24:MI:SS.FF4') FROM RDB$DATABASE;
                                                    -- 2000-12-08 12:35:30.5000
```

3.30 Неявное преобразование типов данных

В РЕД Базе Данных возможно неявное преобразование данных одних типов в другой без применение функции `CAST`. Например, в выражении отбора можно записать:

```
WHERE DOC_DATE < '31.08.2014'
```

и преобразование строки в дату произойдёт неявно.

При конкатенации множества элементов разных типов, все не строковые данные будут неявно преобразованы к строке, если это возможно.

```
select 30|| ' days hath September,April,June and November' CONCAT$ FROM rdb$database;

CONCAT$
=======================================================
30 days hath September,April,June and November
```

Однако, при выполнении арифметических операций (сложение, вычитание и др.) в 3 диалекте требуется указывать функцию `CAST` для явной трансляции одного типа в другой. В 1 диалекте все символьные данные будут неявно преобразованы в число, если это возможно. Например:

```
2 + '1'
```

корректно выполнится. В 3-м диалекте подобное выражение вызовет ошибку, в нем потребуется запись следующего вида:

```
2 + CAST('1' AS SMALLINT)
```

Глава 4

JSON

4.1 Введение в SQL/JSON

СУБД РЕД База Данных 5 поддерживает утвержденный ISO стандарт SQL/JSON, который описывает функционал для работы с JSON данными из SQL.

`JSON` — это текстовый формат хранения данных, который состоит из ключевых элементов и их комбинаций:

- Объект (пара или набор пар в формате `ключ:значение`);

```
{"id": 32}
```

- Массив (список значений, заключенных в квадратные скобки);

```
[10, 3, 45, 4, 5]
```

- Скаляр (число, строка, заключенная в двойные кавычки, логическое значение);
- Значения `null`.

В качестве элементов массива и значений объектов можно использовать другие массивы и объекты.

В приведенном ниже примере JSON-текст представляет собой объект, который содержит массив и строковое значение:

```
{
   "RESPONSE": [{"CODE": 404},
                {"CODE": 418},
                null],
   "ELEMENT": "BUTTON"
}
```

Функционал SQL/JSON включает различные функции для генерации, запроса и валидации данных в JSON:

Функция	**Запроса**		**Генерации**		**Валидации**
	`JSON_VALUE` `JSON_TABLE`	`JSON_QUERY` `JSON_MODIFY`	`JSON_ARRAY` `JSON_OBJECT`	`JSON_ARRAYAGG` `JSON_OBJECTAGG`	`IS JSON` `JSON_EXIST`
Принимает	JSON-текст и JSON-путь		Набор любых аргументов	Селективный столбец	JSON-текст
Возвращает	`VARCHAR`	`VARCHAR (JSON)`			`BOOLEAN`

Функции генерации используются для создания JSON данных (объектов или массивов) на основе значений SQL. *Функции запроса* позволяют извлекать данные из JSON по выражению пути. Из результата убираются пробелы, переносы на новую строку и символы табуляции. *Функции валидации* используются для проверки правильности JSON данных.

4.2 Язык путей JSON

Для получения данных из объекта в SQL JSON используется *Язык путей JSON*. В пути указывается, к какому именно элементу нужно получить доступ. Например, `'$.ID'` обратится к элементу `JSON` с ключом `ID`.

Общий синтаксис языка путей:

```
<выражение пути> ::= '<режим> <путь>'

<путь> ::=  $[<способ доступа>...] [<метод элемента>] [<выражение фильтра>]

<способ доступа> ::=  <доступ к элементу объекта>
                    | <доступ к элементу массива>
```

Язык путей `SQL/JSON` чувствителен к регистру как в идентификаторах, так и в ключевых словах, нет автоматического преобразования идентификаторов в верхний регистр.

Выражение пути является строкой, поэтому его необходимо заключить в одинарные кавычки. Внутри этой строки можно написать строковый литерал, заключив его в двойные кавычки. Апостроф внутри двойных кавычек нужно экранировать. Это можно сделать, используя либо соглашение `SQL` о написании символа дважды, либо экранирование `JavaScript`.

- Экранирование методом написания дважды:

```
'$."Name''"'
```

  Кавычки в примере интерпретируются следующим образом:
  - Внешние одинарные кавычки содержат выражение пути;
  - Двойные кавычки содержат символьную строку на языке пути;
  - Внутренний апостроф экранируется, поскольку содержится в строковом литерале. Фактически обозначает один символ.
- Экранирование по методу `JavaScript`, то есть `\'`:

```
'$."Name\'"'
```

- Экранирование методом записывания символа в `unicode`, например:

```
'$."Name\ u0027"'
```

- Экранирование двойных кавычек:

```
'$."\"Name"'
```

4.2.1 Режимы: lax и strict

Механизм пути имеет два режима: `lax` и `strict`.

На данный момент режим `strict` не проверяет соответствие `JSON` указанному `JSON Path`.

Объявление режима:

```
<выражение пути> ::= <режим> <путь>
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
<режим> ::=  strict | lax
```

По умолчанию используется режим `lax`.

Режим `lax` преобразует ошибки в пустые последовательности `SQL/JSON`.

Режимы пути регулируют три аспекта: разворачивание массивов, оборачивание элементов в массив и обработку ошибок.

В режиме `lax` массив, содержащий только один элемент, является взаимозаменяемым с этим значением, например, `["hello"]` эквивалентно `"Hello"`. Это подкрепляется следующими правилами:

- Если для операции требуется массив, но операнд не является массивом, то операнд оборачивается в массив;
- Если для операции требуется не массив, но операнд является массивом, то операнд разворачивается в последовательность.

Разворачивание массивов:

- В режиме `lax` – массивы автоматически разворачиваются перед выполнением операции, это означает, что `[*]` можно опустить ;
- В режиме `strict` – массивы не разворачиваются автоматически (в таком случае нужно написать `[*]`, чтобы развернуть массив).

Оборачивание элементов в массив:

- В режиме `lax` – операции пути, применяющиеся к индексу, такие как `$[0]` или `$[*]`, могут быть применены к значению, которое не является массивом, но для этого значение неявно оборачивается в массив перед применением операции;
- В режиме `strict` – автоматическое оборачивание перед выполнением операций, применяющихся к индексу, не выполняется.

Обработка ошибок:

- В режиме `lax` – многие ошибки, связанные с тем, являются ли данные массивом или скаляром, обрабатываются функциями автоматического разворачивания и оборачивания. Остальные ошибки классифицируются либо как структурные, либо как неструктурные. Примером структурной ошибки является `$.name`, если в `$` нет элемента, ключом которого является `name`. Структурные ошибки преобразуются в пустые последовательности `SQL/JSON`. Примером неструктурной ошибки является деление на ноль;
- В режиме `strict` – ошибки строго определены во всех случаях.

4.3 Доступ к элементам

Синтаксис обращения к элементам:

```
<выражение пути> ::= <режим> <путь>

<путь> ::=  <доступ к элементу>
         | [<метод элемента>]
         | [<выражение фильтра>]

<доступ к элементу> ::= <контекстная переменная>[<способ доступа>...]

<способ доступа> ::=  <доступ к элементу объкта>
                    | <обращение к элементу объекта подстановочным знаком>
                    | <доступ к элементу массива>
                    | <обращение к элементу массива подстановочным знаком>
```

4.3.1 Обращение к элементу объекта

Синтаксис обращения к элементу объекта:

```
<доступ к элементу объекта> ::=  .<ключ>
                               | .<строка>
                               | .<*>
```

Есть три способа определения ключа.

Первый способ — написание имени ключа открытым текстом. Применяется, если имя ключа не начинается со знака доллара (`$`) и удовлетворяет правилам идентификатора `JavaScript`. Например:

```
$.name
$.firstName
$.Phone
```

Второй способ — написание имени ключа строкой. Это нужно, когда имя ключа начинается со знака доллара (`$`) или содержит специальные символы. Например:

```
$."name"
$."$price"
$."home address"
```

Третий способ — обращение к элементу объекта с помощью подстановочного символа. Например, `*` запросит все поля объекта:

```
$.*
```

4.3.2 Обращение к элементу массива

```
<доступ к элементу массива> ::= [ <список индексов> ]

<список индексов> ::= <индекс> [ {, <индекс> }... ]

<индекс> ::=  <диапазон номеров>
            | <номер> to <номер>
            | *

<номер> ::= <число> | last | <числовое выражение>
```

Квадратные скобки содержат список индексов, разделённых запятыми. Список индексов может быть определён двумя способами: числом или диапазоном между двумя числами, указанным с ключевым словом `to`. Также к элементу массива можно обратиться с помощью подстановочного знака `[*]`. Например, `$[*]` развернёт массив в последовательность элементов. В режиме `strict` операнд должен быть массивом. В режиме `lax` операнд, не являющийся массивом, будет обёрнут в массив.

Индексы рассчитываются с 0. Таким образом, `[0]` — это первый элемент в массиве.

Для доступа к последнему элементу массива неизвестного размера можно использовать ключевое слово `last` в качестве индекса. Например, `$[last]` обратится к последнему элементу массива, а `$[last-1 to last]` вернёт два последних элемента массива.

Например:

```
$[0, last-1 to last, 5]
```

В данном случае обратятся к первому элементу массива, двум последним и шестому.

В режиме `lax` выражение `$[*]` аналогично `$[0 to last]`. В режиме `strict` между такими обращениями есть различие: `$[0 to last]` требует, чтобы массив содержал хотя бы один элемент, тогда как `$[*]` не вернёт ошибку, если `$` является пустым массивом.

К элементам массива можно обращаться по отрицательным индексам. Например, `$[-1]` обратится к последнему элементу массива, а `$[-2]` к предпоследнему.

Индексы могут быть указаны в любой последовательности и могут повторяться. В режиме `strict` индекс должен быть числовым значением в диапазоне между 0 и `last`. В режиме `lax` индексы, выходящие за границы массива игнорируются. Нечисловые значения индексов в любом режиме будут считаться ошибкой.

4.4 Методы элементов

Методы элементов — это методы, которые можно использовать в `JSON Path`.

Синтаксис методов элементов:

```
<выражение пути> ::= '<режим> <путь>'

<путь> ::=  $[<доступ к элементу>] [<метод элемента>] [<выражение фильтра>]

<метод элемента> ::= .<метод>

<метод> ::= type()
          | size()
          | double()
          | ceiling()
          | floor()
          | abs()
          | keyvalue()
```

Методы элементов разворачивают массив в режиме `lax`. Исключение — методы `type()` и `size()`, иначе было бы невозможно узнать их тип или размер.

4.4.1 Метод type()

Метод `type()` возвращает строку с именем типа элемента `SQL/JSON`:

- Если элемент – `null`, то вернёт "null";
- Если элемент – `true` или `false`, то вернёт "boolean";
- Если элемент – число, то вернёт "number";
- Если элемент – строка, то вернёт "string";
- Если элемент – массив, то вернёт "array";
- Если элемент – объект, то вернёт "object".

Например, можно вывести только строки:

```
SELECT JSON_QUERY(
  '{"data":[123,"123","words",false,true,null,[],{}]}', '$.* ? (@.type()=="string")'
  RETURNING VARCHAR(100) WITH ARRAY WRAPPER) FROM RDB$DATABASE;
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
==================================================
["123","words"]
```

Или вывести типы элементов:

```
SELECT JSON_QUERY(
  '{"data":[123,"123","words",false,true,null,[],{}]}', '$.data[*].type()'
  RETURNING VARCHAR(100) WITH ARRAY WRAPPER) FROM RDB$DATABASE;

==================================================
["number","string","string","boolean","boolean","null","array","object"]
```

4.4.2 Метод size()

Метод `size()` возвращает размер элемента SQL/JSON:

- Размер массива равен количеству элементов в массиве;
- Размер объекта или скаляра равен 1.

Например, можно вывести массивы, размер которых больше 1:

```
SELECT JSON_QUERY('[[1, 2, 3],[1],[1, 2]]', '$ ? (@.type()=="array" && @.size()>1)'
                  RETURNING VARCHAR(100) WITH ARRAY WRAPPER) FROM RDB$DATABASE;

==================================================
[[1,2,3],[1,2]]
```

Можно посчитать количество элементов в массиве:

```
SELECT JSON_QUERY('{"data":[1, 2, 3, 4, 5, 6, 7, 8, 9]}', '$.data.size()'
                   RETURNING VARCHAR(100) WITH ARRAY WRAPPER) FROM RDB$DATABASE;

==================================================
[9]
```

4.4.3 Числовые методы элементов: double(), ceiling(), floor(), abs()

Числовые методы элементов выполняют общие числовые функции:

- `double()` преобразует строку в приблизительное числовое значение;
- `ceiling()`, `floor()` и `abs()` выполняют те же операции, что и `CEILING`, `FLOOR` и `ABS` в SQL.

Все эти функции при обработке значения `null` возвращают "null". Если числовые методы применяются не к числу (например, к объекту), то результатом будет ошибка, но если присутствует оператор сравнения, то результатом будет "Unknown".

Пример применения метода `double()`:

```
SELECT JSON_VALUE('{"numbers": "555"}', '$.numbers.double()') FROM RDB$DATABASE;

========================
555
```

Пример применения метода `abs()`:

```
SELECT JSON_VALUE('{"numbers": -555.25}', '$.numbers.abs()') FROM RDB$DATABASE;

=========================
555.25
```

Пример применения метода `ceiling()`:

```
SELECT JSON_VALUE('{"numbers": 555.25}', '$.numbers.ceiling()') FROM RDB$DATABASE;

=========================
556
```

Пример применения метода `floor()`:

```
SELECT JSON_VALUE('{"numbers": 555.25'}', '$.numbers.floor()') FROM RDB$DATABASE;

=========================
555
```

4.4.4 Метод keyvalue()

Метод `keyvalue()` преобразовывает поля `JSON-объекта` в последовательность объектов, описывающих поля исходного.

```
SELECT JSON_QUERY(
       '{ "who": "Fred", "what": 64 }', '$.keyvalue()'
       RETURNING VARCHAR WITH ARRAY WRAPPER ERROR ON ERROR) FROM RDB$DATABASE;

=========================
[{"name":"who","value":Fred,"id":1},{"name":"what","value":64,"id":1}]
```

В примере выше на вход подается один `JSON-объект` с двумя полями, а на выходе получается `JSON-последовательность` из двух объектов с тремя полями у каждого. В результирующей последовательности ключами являются:

- `name` - имя ключа во входном объекте;
- `value` - значение ключа;
- `id` - целое число, являющееся уникальным идентификатором входного `JSON-объекта`.

В результирующей `JSON-последовательности` объекты будут в том же порядке, что и поля в исходном объекте, из которых они преобразованы.

В режиме `lax` вывод `keyvalue()` разворачивается:

```
SELECT JSON_QUERY(
       '[{"who":"Fred","what": 64}, {"who":"Fred","what": 64}] ','lax $.keyvalue()'
       RETURNING VARCHAR(100) WITH ARRAY WRAPPER ERROR ON ERROR) FROM RDB$DATABASE;

=========================
[ { name: "who", value: "Fred", id: 1 }, { name: "what", value: 64, id: 1 },
  { name: "who", value: "Moe",  id: 2 }, { name: "how",  value: 22, id: 2 } ]
```

4.5 Арифметические операции в пути

Язык путей JSON поддерживает следующие арифметические операции:

- Унарные: + и -;
- Бинарные: +, -, *, /, %.

Модуль, обозначаемый символом %, использует тот же алгоритм, что и функция MOD в SQL.

4.5.1 Унарный плюс и минус

Унарные операции плюс и минус выполняют итерации по последовательности JSON. Каждый элемент последовательности должен быть числом (в противном случае, даже в режиме lax, будет получена ошибка). В остальном единственной ошибкой является переполнение при выполнении унарного минуса некоторых чисел на границах их диапазона.

Унарные операции выполняются после получения значений из JSON и после выполнения метода. Например:

```
$ = { readings: [15.2, -22.3, 45.9] }
```

Тогда выражение 'lax -$.readings.floor()' эквивалентно lax -($.readings.floor())

В режиме lax унарные операции разворачивают массив.

Таблица 4.2 — Вычисление lax -$.readings.floor()

Шаг	Выражение	Значение
1	$	{ readings: [15.2, -22.3, 45.9] }
2	$.readings	[15.2, -22.3, 45.9]
3	$.readings.floor()	15, -23, 45
4	-$.readings.floor()	-15, 23, -45

Для получения другого порядка вычислений необходимо использовать скобки: lax (-$.readings).floor().

Таблица 4.3 — Вычисление lax (-$.readings).floor()

Шаг	Выражение	Значение
1	$	{ readings: [15.2, -22.3, 45.9] }
2	$.readings	[15.2, -22.3, 45.9]
3	-$.readings	-15.2, 22.3, -45.9
4	(-$.readings)	-15.2, 22.3, -45.9
5	(-$.readings).floor()	-16, 22, -46

В режиме strict эти примеры требуют явного [*] для раскрытия массива:

```
strict -$.readings[*].floor()
```

или

```
strict (-$.readings[*]).floor()
```

4.5.2 Бинарные операции

Бинарные операции не выполняют итерации по последовательности JSON. Они требуют, чтобы их операнд был числом, иначе результатом будет ошибка, даже в режиме lax. Бинарные операторы имеют тот же приоритет, что и в SQL.

Пример сложения:

```
SELECT JSON_QUERY(
       '{"digits": [15.2, -22, 45, 0]}', '$.digits[*]-5.1'
       RETURNING VARCHAR(50) WITH ARRAY WRAPPER ERROR ON ERROR) FROM RDB$DATABASE;

JSON_QUERY
==================================================
[10.1,-27.1,39.9,-5.1]
```

Для изменения порядка выполнения нужно использовать скобки:

```
SELECT JSON_VALUE('{"value": 15}', '(-$.value)+2*3-15/5%2'
                  RETURNING VARCHAR(20) ERROR ON ERROR) FROM RDB$DATABASE;

==============
-10
```

```
SELECT JSON_VALUE('{"value": 15}', '-($.value+2*3-15/5%2)'
                  RETURNING VARCHAR(20) ERROR ON ERROR) FROM RDB$DATABASE;

==============
-20
```

4.6 Фильтры

Выражение фильтра аналогично предложению WHERE в SQL — оно используется для нахождения элементов, удовлетворяющих определённому условию.

Синтаксис выражения фильтра:

```
<выражение пути> ::= '<режим> <путь>'

<путь> ::=  $[<способ доступа>...] [<метод элемента>] [<выражение фильтра>]

<выражение фильтра> ::= ? ( <предикат> )

<предикат> ::= $[<способ доступа>...] [<метод элемента>]
             | @[<способ доступа>...] [<метод элемента>]
             | (<предикат>)
             | <логический оператор>  <предикат>
             | <оператор сравнения>   <предикат>
             | <оператор exists>
             | <оператор like_regex>
             | <оператор starts with>
             | <оператор is unknown>
```

При использовании фильтра выполняются следующие действия:

- В режиме `lax` массивы в операнде разворачиваются;
- Предикат вычисляется для каждого элемента в последовательности;
- В результат попадают элементы, для которых предикат принял значение `True`.

Переменная `@` в фильтре используется для обозначения текущего элемента в последовательности.

Язык путей содержит следующие предикаты:

- `exists`, чтобы проверить, получает ли выражение пути непустой результат;
- Операторы сравнения `==, !=, <>, <, <=, >, и >=`;
- `like_regex` для сопоставления значения выражения с шаблоном;
- `starts with` чтобы проверить, является ли второй операнд началом первого операнда;
- `is unknown`, чтобы проверить, является ли результат сравнения значением `unknown`. Предикат возвращает `true` или `false`.

4.6.1 Обработка ошибок в фильтрах

Ошибки могут возникать в выражениях фильтра по двум причинам:

- Если операнды являются выражениями, то вычисление выражений может привести к возникновению ошибки. При вычислении выражений режим `lax` преобразует структурные ошибки в пустую последовательность. Неструктурные ошибки рассматриваются как необработанные ошибки. В режиме `strict` все ошибки являются необработанными;
- Если после вычисления операнда предикат обнаружил, что значение не подходит. Например, `10 == "ten"`, в операндах нет ошибок, но они несопоставимы, поэтому в этом предикате все еще есть ошибка.

При любом типе ошибки предикат возвращает значение `unknown`.

4.6.2 Логические операторы

Операнды `&&` (логическое И) будут вычисляться в любом случае.

Результаты `&&` представлены в таблице:

	True	**False**	**Unknown**
True	`True`	`False`	`Unknown`
False	`False`	`False`	`False`
Unknown	`Unknown`	`False`	`Unknown`

Аналогично, операнды `||` (логическое ИЛИ) будут вычисляться в любом случае.

Возможные результаты `||` представлены в таблице:

	True	**False**	**Unknown**
True	`True`	`True`	`True`
False	`True`	`False`	`Unknown`
Unknown	`True`	`Unknown`	`Unknown`

Результаты `!` (логическое отрицание):

P	NOT P
True	False
False	True
Unknown	Unknown

4.6.3 Операторы сравнения

Операторы сравнения — это ==, !=, <, <=, >, >=, <>.

В режиме lax операторы сравнения автоматически разворачивают операнды.

В следующей таблице приведены возможные варианты сравнения:

	null	скаляр	массив	объект
null	сравниваемые	сравниваемые	несравниваемые	несравниваемые
скаляр	сравниваемые	сравниваться могут строка со строкой, число с числом, логическое значение с логическим значением	несравниваемые	несравниваемые
массив	несравниваемые	несравниваемые	несравниваемые	несравниваемые
объект	несравниваемые	несравниваемые	несравниваемые	несравниваемые

Сравнение объектов с чем-либо, даже с самими собой, не поддерживается.

Сравнение выполняется с использованием семантики SQL с дополнительным правилом: SQL/JSON null равен SQL/JSON null, не больше и не меньше чего-либо.

Операнды могут быть последовательностями. В таком случае формируется перекрестное вычисление. Каждый элемент в одной последовательности сравнивается с каждым элементом в другой последовательности. Результат unknown, если какая-либо пара элементов несопоставима. Результат true, если любая пара элементов сравнивается и результат сравнения удовлетворяет условию оператора. Во всех остальных случаях результат false. В режиме lax обработчик пути досрочно прекращает вычисление, если обнаруживает ошибочный или успешный результат. В режиме strict обработчик пути должен проверять все сравнения, и если какое-либо из этих сравнений несопоставимо, то результатом будет unknown.

4.6.4 Предикат like_regex

Предикат сравнивает выражение символьного типа с регулярным выражением.

Синтаксис предиката like_regex:

```
<предикат like_regex> ::= <предикат> like_regex "<регулярное выражение>"
                      [ FLAGS "<флаг>[ <флаг>...]" ]

<флаг> ::=  i | m | s | u | t
```

Для регулярных выражений используется синтаксис google re2.

Флаги можно указывать как в верхнем, так и в нижнем регистре. Описание флагов:

- i - чувствительность к регистру (по умолчание не чувствительны);

- `m` - многострочный режим: `\^` и `$` соответствуют началу и концу строки (по умолчанию выключено);
- `s` - символ точки (`.`) соответствует `\ n` (по умолчанию выключено);
- `u` - меняет местами значения `x*` и `x*?`, `x+` и `x+?` и т.д. (по умолчанию выключено);
- `t` - убрать пробелы справа.

Например:

```
SELECT JSON_EXISTS('{"name": "Isaac Asimov"}', '$ ? (@.name like_regex "Asimov")')
FROM RDB$DATABASE;

==============================
<true>
```

4.6.5 Предикат starts with

Предикат `start with` проверяет, является ли второй операнд началом первого операнда.

Синтаксис предиката `start with`:

```
<предикат starts with> ::= <текстовое значение> starts with <искомая строка>

<текстовое значение> ::= <переменная пути>
                       | <переменная passing>
                       | <скалярное значение>

<искомая строка> ::= <текстовое значение>
```

Например:

```
SELECT JSON_EXISTS('{"name": "Isaac Asimov"}', '$ ? (@.name starts with "Isa")')
FROM RDB$DATABASE;

==============================
<true>
```

4.6.6 Предикат exists

Предикат `exists` проверяет, содержит ли результат выражения пути хотя бы один элемент.

Синтаксис предиката `exists`:

```
<предикат exists> ::= exists (<выражение пути>)
```

Если результат выражения пути — ошибка, то предикат `exists` вернёт `Unknown`. Если результат выражения пути — пустая последовательность, то предикат `exists` вернёт `False`. В любом другом случае результатом будет `True`.

Например:

```
SELECT JSON_QUERY('{"data": [1, 2, 3]}', '$ ? (exists (@.data))') FROM RDB$DATABASE;

==================================
{"data":[1,2,3]}
```

4.6.7 Предикат is unknown

Предикат `is unknown` ппроверяет, является ли результат выражение типом `Unknown`.

Синтаксис предиката `is unknown`:

```
<предикат is unknown> ::= (<выражение пути>) is unknown
```

Например:

```
SELECT JSON_EXISTS('{"digits": [1, 2, 3, 4, 5]}', '$.digits ? ((@ < 2) is unknown)')
FROM RDB$DATABASE;

================================================
<false>
```

```
SELECT JSON_EXISTS('{"digits": [1, 2, 3, 4, 5]}', '$.digits ?(("hi">42) is unknown)')
FROM RDB$DATABASE;

JSON_EXISTS
===========
<true>
```

4.7 Функции SQL/JSON

Функции `SQL/JSON` разделены на три группы: функции генерации данных (`JSON_OBJECT`, `JSON_OBJECTTAG`, `JSON_ARRAY` и `JSON_ARRAYAGG`), функции работы с данными (`JSON_VALUE`, `JSON_TABLE`, `JSON_MODIFY` и `JSON_QUERY`) и функции валидации данных `JSON_EXISTS` и `IS JSON`. Функции генерации данных используют значения `SQL` и создают объекты `JSON` или массивы `JSON`, представленные в символьных типах `SQL`. Функции работы с данными используются для выборки `значений JSON` по выражению пути. Из результата убираются пробелы, переносы на новую строку и символы табуляции. Функции валидации проверяют существующий `JSON` на соответствие указанным критериям.

4.7.1 Общий синтаксис функций работы с данными

Всем функциям работы с данными требуется выражение пути, значение `JSON`, а также необязательные значения параметров.

Функции работы с данными используют общий синтаксис:

```
<общий синтаксис> ::= <значение JSON>,
                       <выражение пути>
                      [<оператор передачи контекстных переменных>]
                      [<выходное значение>]

<значение JSON> ::= <значение> [FORMAT <формат>]

<формат> ::= AUTO
           | SQL
           | JSON

<выражение пути> ::= '<режим> <путь>'
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
<режим> ::= strict | lax

<путь> ::=   $[<способ доступа>...] [<метод элемента>] [<выражение фильтра>]

<оператор передачи контекстных переменных> ::=
         PASSING <параметр JSON> [ {, <параметр JSON> } ]

<параметр JSON> ::= <значение и формат> AS <идентификатор>

<выходное значение> ::= RETURNING <тип данных> [ FORMAT <JSON представление> ]

<JSON представление> :=
  SQL
| JSON
```

Выходное значение нельзя указывать для JSON_VALUE и функций валидации (JSON_EXISTS, IS JSON).

4.7.2 Значение JSON

Значение JSON может быть любым типом данных.

```
<значение JSON> ::= <значение> [FORMAT <формат>]

<формат> ::= AUTO
           | SQL
           | JSON
```

Формат может принимать три значения: AUTO, SQL и JSON. По умолчанию используется FORMAT AUTO.

- FORMAT AUTO — это формат, при котором все значения, созданные функциями генерации данных (JSON_OBJECT, JSON_OBJECTAGG, JSON_ARRAY, JSON_ARRAYAGG) или являющиеся результатом работы функции JSON_QUERY, будут обрабатываться как JSON. Все остальные значения будут преобразованы в **JSON скаляр**. Двойные кавычки и слеши будут экранированы.
- FORMAT SQL — формат, при котором указанное значение будет преобразовано в **JSON скаляр**, даже если фактически значение является JSON.
- FORMAT JSON — формат, при котором указанное значение будет обрабатываться как JSON, даже если фактически значение является строкой.

4.7.3 Выражение пути

Выражение пути определяет, какие значения из исходных данных нужно обработать.

```
<выражение пути> ::= <режим> <путь>

<путь> ::=  $[<способ доступа>...] [<метод элемента>] [<выражение фильтра>]
```

Например, можно использовать значения, которые больше 4:

```
SELECT JSON_QUERY(
   '[{"value":4},{"value":6},{"value":42}]','lax $.value ? (@>4)' WITH ARRAY WRAPPER)
FROM RDB$DATABASE;

========================================
[6,42]
```

4.7.4 Оператор передачи контекстных переменных

Оператор PASSING используется для передачи параметров в выражение пути. Синтаксически оператор PASSING представляет собой список значений с псевдонимом для каждого:

```
<оператор передачи контекстных переменных> ::=
         PASSING <параметр JSON> [ , <параметр JSON> ... ]

<параметр JSON> ::= <значение> [FORMAT <формат>] AS <идентификатор>

<формат> ::= AUTO | SQL | JSON
```

<Идентификатор> является именем переменной, с помощью которого на значение можно ссылаться в выражении пути.

```
SELECT JSON_QUERY(
   '[{"value":4},{"value":6},{"value":42}]', 'lax $.value ? (@>$TR)' PASSING 5 AS TR
   RETURNING VARCHAR(100) WITH ARRAY WRAPPER) FROM RDB$DATABASE;

========================================
[6,42]
```

В примере число 5 передаётся в качестве параметра с помощью переменной TR.

4.7.5 Выходное значение

Выходное значение нельзя указывать для JSON_VALUE и функций валидации (JSON_EXISTS, IS JSON).

Синтаксис предназначен для указания типа данных JSON-текста, возвращающегося JSON функцией:

```
<выходное значение> ::= RETURNING <тип данных> [ FORMAT <JSON представление> ]

<JSON представление> :=
  SQL
| JSON
```

Если возвращаемый тип данных не задан, то по умолчанию используется тип входного значения:

- если входное значение типа BLOB, то выходное значение будет BLOB TEXT в кодировке UTF8;
- если входное значение текстового типа, то выходным значением будет VARCHAR размером длина входной строки + 2 в кодировке UTF8.

- если входное текстовое значение больше максимального размера VARCAHR, то для выходного значения будет использован тип BLOB SUB_TYPE TEXT в кодировке UTF8.
- если входной значение не является текстовым типом, то выходным значением будет VARCAHR достаточного размера. Если не получится вычислить достаточный размер строки, то будет использован тип BLOB SUB_TYPE TEXT в кодировке UTF8.

Если опциональное предложение FORMAT не указано, то по умолчанию предполагается FORMAT JSON. При использовании FORMAT JSON входная строка будет обрабатываться как JSON, даже если фактически является строкой. Строка в этом случае будет экранироваться.

```
SELECT JSON_QUERY('[]', '$' RETURNING VARCHAR(30) FORMAT JSON) IS JSON FORMAT
FROM RDB$DATABASE;

========
<true>
```

4.8 Функции работы с данными

Функции работы с данными:

- JSON_VALUE — извлекает скалярное значение;
- JSON_QUERY — извлекает объекты JSON и массивы JSON;
- JSON_TABLE — извлекает реляционные данные из данных JSON;
- JSON_MODIFY — изменяет существующие значения или добавляет новые.

4.8.1 JSON_VALUE

JSON_VALUE — это функция для извлечения скалярного значения SQL JSON.

Синтаксис оператора:

```
JSON_VALUE ( <значение JSON>,
             <выражение пути>
             [<оператор передачи контекстных переменных>]
             [<выходное значение>]
             [<поведение при пустом значении> ON EMPTY ]
             [<поведение при ошибке> ON ERROR ]
           )

<поведение при пустом значении> ::= ERROR
                                  | NULL
                                  | DEFAULT <пользовательское значение>

<поведение при ошибке> ::= ERROR
                         | NULL
                         | DEFAULT <пользовательское значение>
```

Функция JSON_VALUE может извлекать только одно значение из JSON. Если есть разворачивание и в массиве один элемент, то функция вернёт его без ошибки:

```
SELECT JSON_VALUE('{"numbers": [555.25]}', '$.numbers.abs()') FROM RDB$DATABASE;

========================================
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
555.25
```

Подробнее авторазворачивание описано в пункте 4.2.1.

<Поведение при пустом значении> определяет поведение, если результат выражения пути пустой:

- NULL ON EMPTY означает, что JSON_VALUE вернёт значение null;
- ERROR ON EMPTY означает, что будет показана ошибка;
- DEFAULT <пользовательское значение> ON EMPTY означает, что JSON_VALUE вернёт указанное значение.

Для ON EMPTY и ON ERROR по умолчанию используется NULL.

4.8.2 JSON_QUERY

JSON_QUERY — это функция для извлечения массива JSON или объекта JSON.

Синтаксис JSON_QUERY:

```
JSON_QUERY ( <значение JSON>,
             <выражение пути>
             [<оператор передачи контекстных переменных>]
             [<выходное значение>]
             [<разворачивание> WRAPPER ]
             [<отображение кавычек> QUOTES [ ON SCALAR STRING ] ]
             [<поведение при пустом значении> ON EMPTY ]
             [<поведение при ошибке> ON ERROR ]
           )

<разворачивание> ::= WITHOUT [ ARRAY ]
                   | WITH [ CONDITIONAL | UNCONDITIONAL ] [ ARRAY ]

<отображение кавычек> ::= KEEP | OMIT

<поведение при пустом значении> ::= ERROR
                                  | NULL
                                  | EMPTY ARRAY
                                  | EMPTY OBJECT

<поведение при ошибке> ::= ERROR
                         | NULL
                         | EMPTY ARRAY
                         | EMPTY OBJECT
```

Предложения ON EMPTY и ON ERROR те же, что в JSON_VALUE. Опция DEFAULT отсутствует, но можно указать EMPTY ARRAY или EMPTY OBJECT в качестве результата.

Поведение при пустом значении по умолчанию установлено NULL ON EMPTY. Поведение при ошибке по умолчанию NULL ON ERROR. Для разворачивания значением по умолчанию является WITHOUT ARRAY. Если входное значение является значением null, то результатом JSON_QUERY будет null.

При использовании WITH с опцией UNCONDITIONAL результат всегда будет заключаться в квадратные скобки. При использовании WITH с опцией CONDITIONAL результат будет заключаться в квадратные скобки, только если возвращаемое функцией значение не является массивом. Для WITH значением по умолчанию является UNCONDITIONAL.

Если в запросе содержится выражение фильтра (знак вопроса) или числовые методы элементов

(`double()`, `ceiling()`, `floor()`, `abs()`), то массив автоматически разворачивается. С методами `size()` и `type()` массив автоматически разворачиваться не будет.

Если на выходе у `JSON_QUERY` получается последовательность элментов (например, после разврачивания массива), то необходимо добавить предикат `WITH ARRAY WRAPPER`, так как элементы уже являются отдельными значениями. Также `WRAPPER` требуется в любом запросе с квадратными скобками, иначе будет ошибка.

```
SELECT JSON_QUERY(
'{"numbers":["555","345.567","0.12355"]}','$.numbers[*].double()' WITH ARRAY WRAPPER)
FROM RDB$DATABASE;

==========================
[555,345.567,0.12355]
```

4.8.3 JSON_MODIFY

`JSON_MODIFY` — функция позволяет изменить существующий `JSON`: вставить новые поля в объект, добавить элементы в массив, изменить поля/элементы массива или удалить их.

```
JSON_MODIFY ( <значение JSON>,
              <выражение пути>,
               {   DELETE
                 | UPDATE <новое значение>
                 | APPEND <новое значение>
                 | INSERT <новое значение>
               }
               [<выходное значение>]
               [<поведение при пустом значении> ON EMPTY]
               [<поведение при ошибке> ON ERROR ]
            )

<новое значение> ::= <значение> [FORMAT <формат>]

<формат> ::= AUTO | SQL | JSON

<поведение при ошибке> = NULL
                       | ERROR
                       | EMPTY ARRAY
                       | EMPTY OBJECT
                       | DEFAULT <пользовательское значение>

<поведение при пустом значении> = WORK
                                | ERROR
                                | NULL
                                | EMPTY ARRAY
                                | EMPTY OBJECT
                                | DEFAULT <пользовательское значение>
```

Есть несколько режимов работы функции:

- `UPDATE` - используется для замены значения;
- `APPEND` - используется для добавления нового элемента в конец массива;
- `DELETE` - используется для удаления элемента;

- INSERT - для вставки элемента на определённую позицию в массиве.

Режимом работы по умолчанию является UPDATE. При использовании его в режиме lax и указанном WORK ON EMPTY в случае, если по указанному пути значение не найдено, то оно будет добавлено.

```
SELECT JSON_MODIFY('{"data": "test"}', '$.id', UPDATE 5 RETURNING VARCHAR(30))
FROM RDB$DATABASE;

JSON_MODIFY
========================
{"data":"test","id":5}
```

При использовании режима DELETE нужно опустить MODIFY_VALUE и запятую, то есть передавать только два аргумента.

При использовании INSERT значение будет вставлено именно на указанный индекс, а остальные значения сдвинутся. Например:

```
SELECT JSON_MODIFY('[1,2,3,4]', '$[0]', INSERT 'HI' returning varchar(30))
FROM RDB$DATABASE;

========================
["HI",1,2,3,4]
```

При использовании DELETE значение будет удалено:

```
SELECT JSON_MODIFY('{"data": "test", "id":14}', '$.id', delete)
FROM RDB$DATABASE;

JSON_MODIFY
============================
{"data":"test"}
```

<Поведение при ошибке> определяет поведение, если при обработке выражения пути возникла ошибка:

- NULL ON ERROR означает, что при возникновении ошибки JSON_MODIFY вернёт значение null;
- ERROR ON ERROR означает, что при возникновении ошибки будет показана ошибка;
- EMPTY ARRAY ON ERROR означает, что при возникновении ошибки JSON_MODIFY вернёт пустой массив;
- EMPTY OBJECT ON ERROR означает, что при возникновении ошибки JSON_MODIFY вернёт пустой объект;
- DEFAULT <пользовательское значение> ON ERROR означает, что епри возникновении ошибки JSON_MODIFY вернёт указанное значение.

По умолчанию используется NULL ON ERROR.

<Поведение при пустом значении> определяет поведение, если результат выражения пути пустой:

- WORK ON EMPTY означает, что если запрос возвращает пустое значение, то JSON_MODIFY вернёт отформатированный JSON;
- ERROR ON EMPTY означает, что если запрос возвращает пустое значение, то будет показана ошибка;
- NULL ON EMPTY означает, что если запрос возвращает пустое значение, то JSON_MODIFY вернёт значение null;
- ERROR ON EMPTY означает, что при возникновении ошибки будет показана ошибка;
- EMPTY ARRAY ON EMPTY означает, что JSON_MODIFY вернёт пустой массив;

- DEFAULT <пользовательское значение> ON EMPTY означает, что если запрос возвращает пустое значение, то JSON_MODIFY вернёт указанное значение.

По умолчанию используется WORK ON EMPTY.

4.8.4 JSON_TABLE

JSON_TABLE - это функция, которая принимает JSON в качестве входных данных и извлекает из них реляционные данные. Функция имеет три параметра:

- Значение JSON;
- Выражение пути SQL/JSON для указания нуля или более строк;
- Предложение COLUMNS для определения формы выходной таблицы.

Синтаксис предложения JSON_TABLE:

```
JSON_TABLE (   <значение JSON>,
               <выражение пути>
              [<оператор передачи контекстных переменных>]
               <предложение COLUMNS>
              [<предложение PLAN>]
              [<поведение при ошибке JSON_TABLE> ON ERROR ]
           )

<предложение COLUMNS> ::= COLUMNS (<определение столбца> [, <определение столбца> ...
])

<определение столбца> ::= <порядковый номер строки>
                        | <определение стандартного столбца>
                        | <определение форматированного столбца>
                        | <вложенные столбцы>

<порядковый номер строки> ::= <название столбца> FOR ORDINALITY

<определение стандартного столбца> ::=
                          <название столбца> <тип данных>
                      [ PATH <путь к столбцу> ]
                      [ <поведение при пустом значении столбца> ON EMPTY ]
                      [ <поведение при ошибке> ON ERROR ]

<поведение при пустом значении столбца> ::= ERROR
                                          | NULL
                                          | DEFAULT <пользовательское значение>

<поведение при ошибке> ::= ERROR
                         | NULL
                         | DEFAULT <пользовательское значение>

<определение форматированного столбца> ::=
        <название столбца> <тип данных>
        FORMAT <представление JSON>
      [ PATH <путь к столбцу> ]
      [ <разворачивание> WRAPPER ]
      [ <отображение кавычек> QUOTES [ ON SCALAR STRING ] ]
      [ <поведение при пустом значении форматированного столбца> ON EMPTY ]
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
       [ <поведение при ошибке форматированного столбца> ON ERROR ]

  <разворачивание> ::= WITHOUT [ ARRAY ]
                     | WITH [ CONDITIONAL | UNCONDITIONAL ] [ ARRAY ]

  <отображение кавычек> ::=  KEEP | OMIT

  <поведение при пустом значении форматированного столбца> ::= ERROR
                                                             | NULL
                                                             | EMPTY ARRAY
                                                             | EMPTY OBJECT

  <поведение при ошибке форматированного столбца> ::= ERROR
                                                    | NULL
                                                    | EMPTY ARRAY
                                                    | EMPTY OBJECT

  <поведение при ошибке JSON_TABLE> ::= ERROR
                                      | EMPTY
```

Таблица для примера:

Таблица 4.8 — Таблица BOOKCLUB

ID	JCOL
111	{ "Name": "John Smith", "address": { "streetAddress": "21 2nd Street", "city": "New York", "state": "NY", "postalCode": 10021 }, "phoneNumber": [{ "type": "home", "number": "212 555-1234"}, { "type": "fax", "number": "646 555-4567"}] "books": [{ "title": "The Talisman", "authorList": ["Stephen King", "Peter Straub"], "category": ["SciFi", "Novel"] }, { "title": "Far from the Madding Crowd", "authorList": ["Thomas Hardy"], "category": ["Novel"] }] }

(разрыв таблицы)

(разрыв таблицы)

ID	JCOL
222	{ "Name": "Peter Walker", "address": { "streetAddress": "111 Main Street", "city": "San Jose", "state": "CA", "postalCode": 95111 }, "phoneNumber": [{ "type": "home", "number": "408 555-9876"}, { "type": "office", "number": "650 555-2468"}] "books": [{ "title":"Good Omens", "authorList": ["Neil Gaiman", "Terry Pratchett"], "category": ["Fantasy", "Novel"] }, { "title": "Smoke and Mirrors", "authorList": ["Neil Gaiman"], "category": ["Novel"] }] }
333	{ "Name" : "James Lee" }

С помощью предложения `COLUMNS` можно устанавливать четыре типа столбцов: `столбцы с порядковым номером`, `стандартные`, `форматированные` и `вложенные`.

Предложение `FOR ORDINALITY` добавляет столбец с порядковым номером. Нумерация начинается с 1. Номер автоматически увеличивается для каждого нового элемента массива в пределах одного родителя. Для дочерних строк, сформированных предложениями `NESTED PATH`, повторяется номер родительской строки. Если родительский элемент меняется, последовательность сбрасывается.

Пример добавления столбца с порядковым номером:

```
SELECT jt.rowseq, jt.name, jt.book_id, jt.book FROM bookclub,
 JSON_TABLE ( bookclub.jcol, 'lax $'
     COLUMNS ( rowSeq FOR ORDINALITY, name VARCHAR(30) PATH 'lax $.Name',
               NESTED PATH '$.books[*]'
                     COLUMNS (book_id FOR ORDINALITY, book CHAR(30) PATH '$.title'))
 ) AS jt;

     ROWSEQ NAME     BOOK_ID  BOOK
==== ============== ======== ==============================
   1 John Smith     1        The Talisman
   1 John Smith     2        Far from the Madding Crowd
   2 Peter Walker   1        Good Omens
   2 Peter Walker   2        Smoke and Mirrors
   3 James Lee      1        <null>
```

Стандартный столбец создаёт столбцы скалярного типа. Столбец создается с использованием семантики `JSON_VALUE` и использует выражение `JSON` пути. Столбец также содержит необязательные предложения `ON EMPTY` и `ON ERROR` с теми же вариантами выбора и семантикой, что и `JSON_VALUE`.

Форматированный столбец создаёт столбцы скалярного типа. Столбец создается с использованием семантики `JSON_QUERY`. Столбец также содержит необязательные предложения `WRAPPER`, `QUOTES`, `ON EMPTY` и `ON ERROR` с теми же вариантами выбора и семантикой, что и `JSON_QUERY`.

Пример работы JSON_TABLE (таблица BOOKCLUB приведена в таблице 4.8)

```
SELECT jt.postal
FROM bookclub, JSON_TABLE (bookclub.jcol, '$'
                           COLUMNS (postal INT PATH '$.address.postalCode'
                                    DEFAULT '0' ON EMPTY DEFAULT '1' ON ERROR)
                           ) AS jt;

POSTAL
============
10021
95111
0
```

4.8.5 Вложенные столбцы

Последний вариант определения столбца - это вложенные столбцы. Синтаксис представлен ниже:

```
<вложенные столбцы> ::= NESTED [ PATH ] <путь к вложенной таблице>
                        [ AS <имя вложенного пути> ]
                        <предложение COLUMNS>
```

Предложение NESTED позволяет объединить вложение объектов или массивов JSON в одном запросе, а не связывать несколько выражений JSON_TABLE с помощью SQL. За ключевым словом NESTED следует путь, для которого опционально можно указать псевдоним. Путь обеспечивает уточненный контекст для вложенных столбцов. Имя пути в основном используется, чтобы создать явный план.

Пример работы с вложенными столбцами (таблица BOOKCLUB приведена в таблице 4.8)

```
SELECT jt.Name, jt.phone
FROM bookclub, JSON_TABLE ( bookclub.jcol, 'lax $'
                            COLUMNS (Name varchar(50) path '$.Name',
                                     NESTED PATH '$.phoneNumber[*].number'
                                     COLUMNS (phone CHAR(30) PATH '$' NULL ON EMPTY))
                           ) AS jt;

NAME            PHONE
==============  ===============
John Smith      212 555-1234
John Smith      646 555-4567
Peter Walker    408 555-9876
Peter Walker    650 555-2468
James Lee       <null>
```

4.8.6 Предложение PLAN

Для каждого JSON-пути можно указать его псевдоним с попомщью предложения AS. Имена путей являются идентификаторами и должны быть уникальными. Они используются в предложении PLAN для определения плана вывода.

Синтаксис предложения PLAN:

```
<предложение PLAN> ::= PLAN (<план>)
                     | PLAN DEFAULT ( <определение плана по умолчанию> )

<план> ::= <имя пути>
         | <родительский/дочерний>
         | <пара>

<имя пути> ::= <идентификатор>

<родительский/дочерний> ::= <внешний план>
                           | <внутренний план>

<внешний план> ::= <путь> OUTER <первичный план>

<внутренний план> ::= <путь> INNER <первичный план>

<пара> ::=  <план объединения>
          | <перекрестный план>

<план объединения> ::= <первичный план> UNION <первичный план>
                       [ { UNION <первичный план> }... ]

<перекрестный план> ::= <первичный план> CROSS <первичный план>
                        [ { CROSS <первичный план> }... ]

<первичный план> ::=  <имя пути>
                    | ( <план> )

<определение плана по умолчанию> ::=
                  <внутренний/внешний план> [ <план объединения/перекрестный> ]
                | <план объединения/перекрестный> [ , <внутренний/внешний план> ]

<внутренний/внешний план> ::= INNER | OUTER

<план объединения/перекрестный> ::= UNION | CROSS
```

Ключевые слова INNER, OUTER, UNION и CROSS в контексте предложения PLAN имеют следующие характеристики:

- INNER соответствует семантике INNER JOIN.
- OUTER соответствует семантике LEFT OUTER JOIN и используется по умолчанию для отношений родительский/дочерний.
- Первым операндом INNER и OUTER является путь, который должен быть предком всех имен путей во втором операнде.
- Если план явный, то все пути должны быть явными и встречаться в предложении PLAN ровно один раз.
- CROSS соответствует семантике CROSS JOIN.
- UNION соответствует семантике FULL OUTER JOIN неудовлетворительным предикатом, таким как 1=0, и по умолчанию используется для отношений между сиблингами.
- UNION является ассоциативным.
- CROSS является ассоциативным.
- Круглые скобки необходимы для разграничения сложных выражений.

- Не существует приоритета между `UNION` и `CROSS`.

Пример работы плана DEFAULT (таблица `BOOKCLUB` приведена в таблице 4.8)

```
SELECT bookclub.id, jt.name, jt.title, jt.author, jt.category
FROM bookclub,
  JSON_TABLE ( bookclub.jcol, 'lax $'
  COLUMNS (name VARCHAR(30) PATH 'lax $.Name',
           NESTED PATH 'lax $.books[*]'
           COLUMNS (title VARCHAR(60) PATH 'lax $.title',
                    NESTED PATH 'lax $.authorList[*]' AS ATH
                    COLUMNS (author VARCHAR(30) PATH 'lax $'),
                             NESTED PATH 'lax $.category[*]' AS CAT
                             COLUMNS ( category VARCHAR(30) PATH 'lax $')
                   )
          )
  PLAN DEFAULT(INNER,CROSS)
  ) AS jt;

ID  NAME         TITLE              AUTHOR          CATEGORY
=== ============ ================  =============== =========
111 John Smith   The Talisman      Stephen King    SciFi
111 John Smith   The Talisman      Stephen King    Novel
111 John Smith   The Talisman      Peter Straub    SciFi
111 John Smith   The Talisman      Peter Straub    Novel
111 John Smith   Far From the      Thomas Hardy    Novel
                Madding Crowd
222 Peter Walker Good Omens        Neil Gaiman     Fantasy
222 Peter Walker Good Omens        Neil Gaiman     Novel
222 Peter Walker Good Omens        Terry Pratchett Fantasy
222 Peter Walker Good Omens        Terry Pratchett Novel
222 Peter Walker Smoke and Mirrors Neil Gaiman     Novel
```

Пример работы внутреннего плана с объединением

```
SELECT  JT.t, JT.a
FROM JSON_TABLE( '[{"t":[1,2], "a":[10,20]}]', '$[*]' as PERSON
   COLUMNS ( NESTED PATH 'lax $.t[*]' as SUB1 columns(t VARCHAR(30) PATH 'lax $'),
               NESTED PATH 'lax $.a[*]' as SUB2 columns(a VARCHAR(30) PATH 'lax $') )
   PLAN (PERSON INNER (SUB1 UNION SUB2) )
) AS JT;

T         A
========  ==============================
1         <null>
2         <null>
<null>    10
<null>    20
```

Пример работы внутреннего перекрёстного плана

```
SELECT  JT.t, JT.a
FROM JSON_TABLE( '[{"t":[1,2], "a":[10,20]}]', '$[*]' as PERSON
   COLUMNS ( NESTED PATH 'lax $.t[*]' as SUB1 columns(t VARCHAR(30) PATH 'lax $'),
               NESTED PATH 'lax $.a[*]' as SUB2 columns(a VARCHAR(30) PATH 'lax $') )
   PLAN (PERSON INNER (SUB1 CROSS SUB2) )
) AS JT;

T    A
===  ====
1    10
1    20
2    10
2    20
```

Пример работы внешнего плана

```
SELECT  JT.t, JT.a
FROM JSON_TABLE( '{"t":null, "a":[10,20]}', '$' as PERSON
         COLUMNS ( T VARCHAR(30) PATH 'lax $.t',
                  NESTED PATH 'lax $.a[*]' as SUB2
                  COLUMNS ( a VARCHAR(30) PATH 'lax $') )
         PLAN (PERSON OUTER SUB2)
) AS JT;

T        A
=======  ====
<null>   10
<null>   20
```

4.9 Функции валидации

Функции валидации JSON:

- JSON_EXISTS — определяет, существует ли значение по заданному пути;
- IS JSON — определяет, является ли указанное строковое значение текстом JSON.

4.9.1 JSON_EXISTS

Функция JSON_EXISTS необходима для проверки существования какого-либо значения по заданному пути. Возвращает логический результат. True, если выражение пути находит один или несколько элементов SQL/JSON.

Синтаксис оператора:

```
JSON_EXISTS ( <значение JSON>,
              <выражение пути>
              [<оператор передачи контекстных переменных>]
              [<поведение при ошибке> ON ERROR]
            )
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
<поведение при ошибке> ::=   TRUE
                           | FALSE
                           | UNKNOWN
                           | ERROR
```

Пример работы функции:

```
SELECT JSON_EXISTS('{"tags":{"test":[1,2,3,4,5]}}', '$.tags.test[2]')
FROM RDB$DATABASE;

JSON_EXISTS
===========
<true>
```

Предложение `ON ERROR` по умолчанию равно `FALSE`. Если входное значение является значением `null`, то результатом `JSON_EXISTS` будет `Unknown`.

4.9.2 IS JSON

Предикат `IS JSON` проверяет значение на соответствие схеме JSON.

Синтаксис оператора следующий:

```
<предикат IS JSON> ::= <значение> [FORMAT <формат>] IS JSON
                       [ {<тип значения> |  [SYSTEM] FORMAT} ]
                       [ <ограничение уникальности ключа> ]

<тип значения> ::= VALUE
                 | ARRAY
                 | OBJECT
                 | SCALAR

<ограничение уникальности ключа> ::= WITH UNIQUE [ KEYS ]
                                   | WITHOUT UNIQUE [ KEYS ]
```

Например, следующий запрос проверяет, является ли указанное значение `JSON`-текстом:

```
SELECT '[{"value":5}', 10, true]' IS JSON FROM RDB$DATABASE;

========================================
<true>
```

Скалярные значения также являются `JSON`-текстом:

```
SELECT '"String scalar value"' IS JSON FROM RDB$DATABASE;

========================================
<true>
```

`<Тип значения>` позволят проверить, соответствует ли входная строка указанному типу:

- `VALUE` — для проверки, является ли входная строка любым значением `JSON`, в том числе `null`:

```
SELECT 'null' IS JSON VALUE FROM RDB$DATABASE;

====================
<true>
```

При указании SQL значения null результатом будет null:

```
SELECT null IS JSON VALUE FROM RDB$DATABASE;

====================
<null>
```

- ARRAY — для проверки, является ли входная строка массивом:

```
SELECT '[1,2,3]' IS JSON ARRAY FROM RDB$DATABASE;

====================
<true>
```

- OBJECT — для проверки, является ли входная строка объектом:

```
SELECT '{"value":5}' IS JSON OBJECT FROM RDB$DATABASE;

====================
<true>
```

- SCALAR — для проверки, является ли входная строка скаляром.

```
SELECT '1' IS JSON SCALAR FROM RDB$DATABASE;

====================
<true>
```

SYSTEM FORMAT проверяет, является ли входная строка результатом работы JSON-функции:

```
SELECT JSON_QUERY('[]', '$') IS JSON SYSTEM FORMAT FROM RDB$DATABASE;

====================
<true>
```

Типом входных данных по умолчанию является FORMAT JSON.

Ограничение уникальности ключа проверяет, должны ли ключи повторяться. Если есть, то при использовании WITHOUT UNIQUE KEYS функция IS JSON вернёт true, при использовании WITH UNIQUE KEYS — false. По умолчанию используется WITHOUT UNIQUE KEYS.

Работа функции IS JSON с WITHOUT UNIQUE KEYS:

```
SELECT '{"A":1, "B":2, "A":3}' IS JSON FROM RDB$DATABASE;

========================================
<true>
```

Работа функции IS JSON с WITH UNIQUE KEYS:

```
SELECT '{"A":1, "B":2, "A":3}' IS JSON WITH UNIQUE FROM RDB$DATABASE;

========================================
<false>
```

4.10 Функции генерации данных

Функции генерации данных:

- JSON_OBJECT — создаёт объект JSON из явных пар ключ-значение;
- JSON_OBJECTAGG — создаёт объект JSON путем агрегирования полей из таблицы или другого селективного источника;
- JSON_ARRAY — создаёт массив JSON из явного списка данных или подзапроса;
- JSON_ARRAYAGG — создаёт массив JSON путём путём агрегирования данных SQL.

4.10.1 Общий формат входных значений

Функции генерации данных используют общий формат для входных значений:

```
<значение и формат> ::= <значение> FORMAT <формат>
```

<Значение> может быть любым типом данных. Формат может принимать три значения: AUTO, SQL и JSON. По умолчанию используется FORMAT AUTO.

- FORMAT AUTO — это формат, при котором все значения, созданные функциями генерации данных (JSON_OBJECT, JSON_OBJECTAGG, JSON_ARRAY, JSON_ARRAYAGG) или являющиеся результатом работы функции JSON_QUERY, будут обрабатываться как JSON. Все остальные значения будут преобразованы в JSON скаляр. Двойные кавычки и слеши будут экранированы. Например:

```
SELECT JSON_ARRAY('1', '2' FORMAT SQL, '3' FORMAT JSON) from RDB$DATABASE;

============
["1","2",3]
```

- FORMAT SQL — формат, при котором указанное значение будет преобразовано в JSON скаляр, даже если фактически значение является JSON. Например:

```
SELECT JSON_ARRAY(JSON_QUERY('{}', '$'),
                  JSON_QUERY('{}', '$') FORMAT JSON,
                  JSON_QUERY('{}', '$') FORMAT SQL) from RDB$DATABASE;

============
[{},{},"{}"]
```

- FORMAT JSON — формат, при котором указанное значение будет обрабатываться как JSON, даже если фактически значение является строкой. Например:

```
SELECT JSON_ARRAY(1, '2', '3' FORMAT JSON) from RDB$DATABASE;

============
[1,"2",3]
```

4.10.2 JSON_OBJECT

Функция JSON_OBJECT создаёт объекты JSON из явных пар ключ-значение.

Синтаксис функции:

```
JSON_OBJECT ( [<ключ-значение> [, <ключ-значение> ...]]
              [<поведение при значении null>  ON NULL]
              [<ограничение уникальности ключа>]
              [<выходное значение>]
            )

<ключ-значение> ::=  [ KEY ] <ключ JSON> VALUE <значение JSON>
                     | <ключ JSON> : <значение JSON>

<ключ JSON> ::= <строковое значение>

<значение JSON> ::= <значение> FORMAT <формат>

<поведение при значении null> ::= NULL | ABSENT

<ограничение уникальности ключа> ::=  WITH UNIQUE [ KEYS ]
                                    | WITHOUT UNIQUE [ KEYS ]

<выходное значение> ::= RETURNING <тип данных>
```

<Ключ JSON> может быть только строковым типом и не может быть равным NULL. Если <значение JSON> равно NULL, то поведение определяется предложением <поведение при значении null>. NULL ON NULL возврвщает значение NULL, а ABSENT ON NULL опускает такую пару ключ-значение из результирующего объекта. По умолчанию используется NULL ON NULL.

Построение объекта JSON с NULL ON NULL:

```
SELECT JSON_OBJECT('size': 3, key 'name' value null, 'ref': false NULL ON NULL)
FROM RDB$DATABASE;

========================================
{"size":3,"name":null,"ref":false}
```

Построение объекта JSON с ABSENT ON NULL:

```
SELECT JSON_OBJECT('size': 3, key 'name' value null, 'ref': false ABSENT ON NULL)
FROM RDB$DATABASE;

========================================
{"size":3,"ref":false}
```

Ограничение уникальности ключа определяет, должны ли ключи в результате повторяться. При использовании WITHOUT UNIQUE KEYS ключи могут дублироваться. При использовании WITH UNIQUE KEYS ключи должны быть уникальными, иначе результатом работы функции будет ошибка. По умолчанию WITHOUT UNIQUE KEYS.

Работа функции JSON_OBJECT с WITHOUT UNIQUE KEYS:

```
SELECT JSON_OBJECT('A':1, 'B':2, 'A':3) FROM RDB$DATABASE;
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
========================================
{"A":1,"B":2,"A":3}
```

Работа функции JSON_OBJECT с WITH UNIQUE KEYS:

```
SELECT JSON_OBJECT('A':1, 'B':2, 'A':3 WITH UNIQUE) FROM RDB$DATABASE;

======================================
There is a non-unique key with name 'A' on rows 0 and 2
```

4.10.3 JSON_OBJECTAGG

Можно построить объект JSON путем объединения информации из таблицы SQL. Подразумевается, что таблица фактически содержит столбец с ключами JSON и другой столбец с соответствующими ключам значениями.

Синтаксис функции:

```
JSON_OBJECTAGG ( <ключ-значение>
                 [<поведение при значении null> ON NULL]
                 [<ограничение уникальности ключа> ]
                 [<выходное значение> ]
               )

<ключ-значение> ::= <имя столбца с ключами>: <имя столбца со значениями>
                    [FORMAT <формат>]

<поведение при значении null> ::= NULL | ABSENT

<ограничение уникальности ключа> ::= WITH UNIQUE [ KEYS ]
                                   | WITHOUT UNIQUE [ KEYS ]

<выходное значение> ::= RETURNING <тип данных>
```

Пример построения объекта:

```
SELECT JSON_OBJECTAGG(option : id) FROM OPTIONS;

======================================
{"size":1,"name":2,"value":3}
```

Таблица 4.9 — Таблица OPTIONS

ID	OPTION
1	"SIZE"
2	"NAME"
3	"VALUE"

<Поведение при значении null> по умолчанию равно NULL ON NULL.

4.10.4 JSON_ARRAY

Функция JSON_ARRAY позволяет построить массив JSON из явного списка данных.

Синтаксис функции:

```
<конструктор массивов JSON> ::=   <создание массива JSON из списка значений>
                                | <создание массива JSON из результата запроса>

<создание массива JSON из списка значений> ::=
            JSON_ARRAY (
                [ <значение и формат> [, <значение и формат> ... ]
                [ <поведение при значении null>] ON NULL]
                [ <выходное значение> ]
            )

<создание массива JSON из результата запроса> ::=
            JSON_ARRAY (
                 <SQL-запрос SELECT>
               [ FORMAT <формат>]
               [ <поведение при значении null> ON NULL]
               [ <выходное значение> ]
            )

<значение и формат> ::= <значение> [FORMAT <формат>]

<поведение при значении null> ::= NULL | ABSENT

<выходное значение> ::= RETURNING <тип данных>
```

JSON_ARRAY предоставляет два варианта создания массива. Первый вариант создает результат из явного списка значений SQL:

```
SELECT JSON_ARRAY(1,2,3,4,5) FROM RDB$DATABASE;
========================================
[1,2,3,4,5]
```

Второй вариант создаёт массив из результата запроса SQL, вызванного внутри функции. Запрос должен возвращать ровно один столбец, а элементы массива будут сформированы из значений столбца, сгенерированного запросом (таблица OPTION представлена в таблице 4.9):

```
SELECT JSON_ARRAY(select option from options) FROM RDB$DATABASE;
========================================
["size","name","value"]
```

Поведение при значении null по умолчанию равно ABSENT ON NULL (что отличается от значения по умолчанию для JSON_OBJECT).

4.10.5 JSON_ARRAYAGG

Функция JSON_ARRAYAGG позволяет создать массив JSON путём агрегирования данных SQL.

Синтаксис функции:

```
JSON_ARRAYAGG (   <столбец таблицы>
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
                [FORMAT <формат>]
                [<поведение при значении null> ON NULL]
                [<выходное значение>]
            )

<поведение при значении null> ::= NULL | ABSENT

<выходное значение> ::= RETURNING <тип данных>
```

Пример построения массива JSON (таблица OPTION представлена в таблице 4.9):

```
SELECT JSON_ARRAYAGG(option) FROM OPTIONS;

======================================
["size","name","value"]
```

Поведение при значении null по умолчанию равно ABSENT ON NULL.

Глава 5

Транзакции

Все операции в РЕД Базе Данных выполняются в рамках транзакции. Транзакция - логическая единица изолированной работы группы последовательных операций над базой данных. Изменения остаются обратимыми до тех пор, пока клиентское приложение не выдаст серверу инструкцию `COMMIT`.

В РЕД Базе Данных есть небольшое количество SQL-операторов для управления транзакциями:

- `SET TRANSACTION` - определение параметров транзакции и её старт;
- `COMMIT` - завершение транзакции и фиксация изменений;
- `ROLLBACK` - отмена изменений, выполненных в рамках транзакции, или выполняет откат к точке сохранения;
- `SAVEPOINT` - определение точки сохранения для частичного отката изменений;
- `RELEASE SAVEPOINT` - удаление точки сохранения.

5.1 SET TRANSACTION

Оператор `SET TRANSACTION` определяет параметры транзакции и запускает её.

Синтаксис оператора:

Листинг 5.1. Синтаксис оператора запуска транзакции `SET TRANSACTION`

```
 SET TRANSACTION
  [NAME <имя транзакции>]
  [<параметры транзакции> ...]

<параметры транзакции> ::=
  READ {ONLY | WRITE}
| [WAIT [LOCK TIMEOUT <кол-во секунд>] | NO WAIT]
| [ISOLATION LEVEL] <уровень изоляции>
| NO AUTO UNDO
| AUTO COMMIT
| [AUTO RELEASE TEMP BLOBID]
| IGNORE LIMBO
| RESERVING <список таблиц для резервирования>
|  USING <хендл БД> [,<хендл БД>]

<уровень изоляции> ::=
  SNAPSHOT [AT NUMBER <номер снимка транзакции>]
| SNAPSHOT TABLE [STABILITY]
| READ COMMITTED [[NO] RECORD_VERSION | READ CONSISTENCY]

<список таблиц для резервирования> ::= <определение таблицы> [, <определение таблицы>
...]

<определение таблицы> ::= <имя таблицы> [, <имя таблицы> ...]
  [FOR [SHARED | PROTECTED] {READ | WRITE}]
```

Параметры оператора `SET TRANSACTION`:

Таблица 5.1 — Параметры оператора SET TRANSACTION

Имя транзакции	Имя транзакции. Доступно только в ESQL.
Параметры транзакции	Необязательные параметры транзакции. Каждая опция может быть указана не более одного раза, а некоторые опции являются взаимоисключающими (например, READ ONLY и READ WRITE, WAIT и NO WAIT).
Кол-во секунд	Время в секундах, в течение которого оператор будет ждать в случае возникновения конфликта. Должно быть равно 0 или больше.
Номер снимка транзакции	Номер снимка другой транзакции, данные которой должны быть общими с новой транзакцией.
Хендл БД	Хендл базы данных, к которой транзакция может получить доступ. Доступно только в ESQL.

Оператор SET TRANSACTION задаёт параметры транзакции и запускает её. Старт транзакции осуществляется только клиентскими приложениями, но не сервером (за исключением автономных транзакций и некоторых фоновых системных потоков/процессов, например, sweep).

Клиентское приложение может запускать любое количество параллельно выполняющихся транзакций. Одно соединение может иметь несколько одновременных активных транзакций. Фактически есть ограничение на общее количество выполняемых транзакций во всех клиентских приложениях, работающих с одной конкретной базой данных с момента последнего восстановления базы данных из резервной копии или с момента создания базы данных. Это ограничение составляет $2^{48} - 1$, то есть 281474976710656.

Все предложения в операторе SET TRANSACTION являются необязательными. Если предложения не указаны, то предполагается старт транзакции со значениями всех характеристик по умолчанию (режим доступа, режим разрешения блокировок и уровень изоляции).

По умолчанию транзакция запускается со следующими характеристиками:

```
SET TRANSACTION
  READ WRITE
  WAIT
  ISOLATION LEVEL SNAPSHOT;
```

Драйверы баз данных или компоненты доступа могут использовать другие значения по умолчанию для транзакций, запускаемых через их API.

Сервер последовательно присваивает транзакциям целые номера. Каждый раз, когда клиент запускает какую-либо транзакцию, определенную явно или по умолчанию, сервер отправляет клиенту идентификатор транзакции. Этот номер можно узнать с помощью контекстной переменной CURRENT_TRANSACTION.

Некоторые драйверы баз данных требуют, чтобы транзакция настраивалась и запускалась с помощью методов API. В этом случае использование SET TRANSACTION либо не поддерживается, либо может привести к неоднозначному поведению.

5.1.1 Имя транзакции

Необязательное предложение NAME задаёт имя транзакции. Предложение NAME доступно только в Embedded SQL. Если предложение NAME не указано, то оператор SET TRANSACTION применяется к транзакции по умолчанию. За счёт именованных транзакций позволяется одновременный запуск нескольких активных транзакций в одном приложении. При этом должна быть объявлена и инициализирована одноименная переменная базового языка. В DSQL, это ограничение предотвращает динамическую спе-

цификацию имён транзакций.

5.1.2 Параметры транзакции

Основными характеристиками транзакции являются:

- Режим доступа к данным (`READ WRITE, READ ONLY`),
- Режим разрешения блокировок (`WAIT, NO WAIT`) с возможным дополнительным уточнением (`LOCK TIMEOUT`),
- Уровень изоляции (`READ COMMITTED`, `SNAPSHOT`, `SNAPSHOT TABLE STABILITY`),
- средства резервирования или освобождения таблиц (предложение `RESERVING`).

5.1.3 Режим доступа

Для транзакций существует два режима доступа к данным базы данных: `READ WRITE` и `READ ONLY`.

При режиме доступа `READ WRITE` операции в контексте данной транзакции могут быть как операциями чтения, так и операциями изменения данных. Это режим по умолчанию.

В режиме `READ ONLY` в контексте данной транзакции могут выполняться только операции выборки данных `SELECT`. Любая попытка изменения данных в контексте такой транзакции приведет к исключениям базы данных. Однако это не относится к глобальным временным таблицам (`GTT`), которые разрешено изменять в `READ ONLY` транзакциях. Этот режим доступа совместно с некоторыми другими характеристиками базы данных должен использоваться при работе с базами данных только для чтения, находящимися на носителях, допускающих только чтение данных, например, на компакт-дисках. Подробности см. в документе «Руководство администратора».

5.1.4 Режим разрешения блокировок

Когда несколько клиентских процессов работают с одной базой данных, могут возникать блокировки, когда один процесс вносит незафиксированные изменения в строку таблицы или удаляет строку, а другой процесс пытается обновить или удалить ту же строку. Такие блокировки называются конфликтами обновлений. Блокировки могут возникать и в других ситуациях, когда используется несколько уровней изоляции транзакций.

Есть два режима разрешения блокировок: `WAIT` и `NO WAIT`.

В режиме `WAIT` (режим по умолчанию), если конфликт возникает между двумя параллельными процессами, выполняющими одновременное обновление данных в одной и той же базе данных, транзакция `WAIT` будет ждать, пока другая транзакция не завершит работу - зафиксирует (`COMMIT`) или откатится (`ROLLBACK`). Клиентское приложение с транзакцией `WAIT` будет переведено в режим ожидания, пока конфликт не разрешится.

Если для транзакции `WAIT` указано `LOCK TIMEOUT`, ожидание будет продолжаться только в течение указанного в этом пункте количества секунд. Если по истечении указанного интервала блокировка не будет разрешена, клиенту будет возвращено сообщение `«Lock time-out on wait transaction»`.

В зависимости от уровня изоляции транзакции поведение при разрешении блокировки может незначительно отличаться.

В режиме `NO WAIT` при возникновении конфликта транзакция немедленно выбрасывает исключение из базы данных.

`LOCK TIMEOUT` - это отдельная опция, но может использоваться только для транзакций `WAIT`. Указание `LOCK TIMEOUT` дл `NO WAIT` приведет к ошибке `«invalid parameter in transaction parameter block -Option isc_tpb_lock_timeout is not valid if isc_tpb_nowait was used previously in TPB»`.

5.1.5 Уровень изоляции

Уровень изоляции транзакций определяет уровень, при котором в транзакции допускаются несогласованные данные, то есть степень изолированности одной транзакции от другой. Изменения, внесённые некоторым оператором, будут видны всем последующим операторам, запущенным в рамках этой же транзакции, независимо от её уровня изоляции. Изменения, сделанные в рамках другой транзакции, остаются невидимыми для текущей транзакции до тех пор, пока они не подтверждены. Уровень изоляции определяет, как транзакция будет взаимодействовать с другой транзакцией, которая хочет подтвердить изменения.

Параметр `ISOLATION LEVEL` определяет уровень изоляции для запускаемой транзакции. Это наиболее значимый параметр транзакции, определяющий ее поведение по отношению к другим параллельно выполняющимся транзакциям.

В РЕД Базе Данных поддерживаются три уровня изоляции:

- `SNAPSHOT`
- `SNAPSHOT TABLE STABILITY`
- `READ COMMITTED` с тремя уточнениями (`READ CONSISTENCY`, `NO RECORD_VERSION` и `RECORD_VERSION`)

Уровень изоляции SNAPSHOT

С этим уровнем изоляции транзакция запускается по умолчанию. Уровень изоляции `SNAPSHOT` позволяет транзакции видеть только те изменения, которые были зафиксированы до ее запуска. Любые зафиксированные изменения, сделанные параллельными транзакциями, не будут видны транзакции `SNAPSHOT`, пока она активна. Изменения станут видны новой транзакции, когда текущая транзакция будет зафиксирована или откачена, но не в том случае, если это будет только откат к точке сохранения.

Изменения, сделанные автономными транзакциями, не видны в контексте транзакции `SNAPSHOT`, которая их запустила.

Предложение AT NUMBER

Транзакцию с уровнем изолированности `SNAPSHOT` можно запустить на основе другой транзакции, если известен номер её снимка. В этом случае эта новая транзакция может видеть те же самые данные, что и транзакция на основе которой она запущена.

Предложение `AT NUMBER` позволяет создавать параллельные процессы (в разных подключениях), считывающие согласованные данные из базы данных. Например, процесс резервного копирования может создавать несколько потоков, параллельно считывающих данные из базы данных. Или веб-служба работать с распределенными вспомогательными службами, выполняя некоторую обработку.

Для этого используется следующий синтаксис:

```
SET TRANSACTION SNAPSHOT AT NUMBER <номер снимка транзакции>
```

`<номер снимка транзакции>` можно получить используя:

```
select RDB$GET_CONTEXT('SYSTEM', 'SNAPSHOT_NUMBER') from rdb$database;
```

Для обмена стабильным снимком между транзакциями другая транзакция также должна иметь уровень изоляции `SNAPSHOT`. При выполнении `READ COMMITTED` номер моментального снимка переместится вперед.

Уровень изоляции SNAPSHOT TABLE STABILITY

Уровень изоляции SNAPSHOT TABLE STABILITY (или SNAPSHOT TABLE) является самым строгим. Как и в SNAPSHOT, транзакция SNAPSHOT TABLE STABILITY видит только те изменения, которые были зафиксированы до начала текущей транзакции. После запуска SNAPSHOT TABLE STABILITY никакие другие транзакции не могут вносить изменения в любую таблицу в базе данных, в которой есть изменения, ожидающие этой транзакции. Другие транзакции могут читать другие данные, но любая попытка вставки, обновления или удаления параллельным процессом приведет к возникновению конфликтных исключений.

Чтобы разрешить другим транзакциям изменять данные в некоторых таблицах, можно использовать предложение RESERVING. Если любая другая транзакция ожидает незафиксированного изменения в любой (не SHARED) таблице, перечисленной в предложении RESERVING, попытка запустить транзакцию SNAPSHOT TABLE STABILITY приведет к неопределенному ожиданию (по умолчанию или явному WAIT) или исключению (NO WAIT или по истечении LOCK TIMEOUT).

Для таблиц, указанных в предложении RESERVING, в параллельных транзакциях в зависимости от их уровня изоляции допустимы при различных способах резервирования следующие варианты поведения:

- SHARED READ — позволяет всем параллельным транзакциям независимо от их уровня изоляции не только читать, но и выполнять любые изменения в резервируемых таблицах (если параллельная транзакция имеет режим доступа READ WRITE);
- SHARED WRITE — для всех параллельных транзакций с уровнем доступа READ WRITE и с уровнями изоляции SNAPSHOT и READ COMMITTED позволяет читать данные из таблиц и писать данные в указанные таблицы, для транзакций с уровнем изоляции SNAPSHOT TABLE STABILITY запрещает не только запись, но также и чтение данных из указанных таблиц;
- PROTECTED READ — допускает только лишь чтение данных из резервируемых таблиц для параллельных транзакций с любым уровнем изоляции;
- PROTECTED WRITE — для параллельных транзакций с уровнями изоляции SNAPSHOT и READ COMMITTED запрещает запись в указанные таблицы, для транзакций с уровнем изоляции SNAPSHOT TABLE STABILITY запрещает также и чтение данных из резервируемых таблиц.

Уровень изоляции READ COMMITTED

Уровень изоляции READ COMMITTED позволяет текущей незафиксированной транзакции сразу же увидеть все изменения данных, которые были зафиксированы другими транзакциями с момента ее запуска. Нефиксированные изменения не видны транзакции READ COMMITTED.

Чтобы получить обновленный набор строк в интересующей таблице, необходимо снова выполнить оператор SELECT, находясь все еще в незафиксированной транзакции READ COMMITTED.

Опции READ COMMITTED

Для транзакций READ COMMITTED может быть задан один из трех уточняющих параметров, в зависимости от желаемого типа разрешения конфликтов: READ CONSISTENCY, RECORD_VERSION или NO RECORD_VERSION. Если в firebird.conf или в databases.conf параметр ReadConsistency установлен в 1, указанные опции фактически игнорируются и ведут себя как READ CONSISTENCY. В противном случае эти опции являются взаимоисключающими.

Опции READ COMMITTED:

- NO RECORD_VERSION (по умолчанию, если ReadConsistency = 0) - это механизм двухфазной блокировки: он делает транзакцию неспособной записывать в любую строку, в которой ожидается обновление от другой транзакции.
  - Если указано NO WAIT, то транзакция немедленно выдаст ошибку о конфликте блокировок.
  - Если указано WAIT, она будет ждать, пока другая транзакция не будет либо за-

фиксирована, либо отменена. Если другая транзакция откатывается, или если она зафиксирована и ее идентификатор транзакции старше идентификатора текущей транзакции, то изменение текущей транзакции разрешается. Если другая транзакция была зафиксирована и ее идентификатор был новее, чем у текущей транзакции, возвращается ошибка конфликта блокировок.

- Если указано значение RECORD_VERSION, транзакция считывает последнюю зафиксированную версию строки, независимо от других ожидающих версий строки. Режим разрешения блокировки (WAIT или NO WAIT) никак не влияет на поведение транзакции при ее старте.
- Если указано READ CONSISTENCY (или ReadConsistency = 1), при выполнении оператора создается снимок базы данных, чтобы обеспечить последовательное чтение на уровне операторов транзакций, зафиксированных на момент начала выполнения.

Два других варианта могут привести к непоследовательным чтениям на уровне оператора, поскольку они могут считывать некоторые, но не все изменения параллельной транзакции, если эта транзакция фиксируется во время выполнения оператора. Например, оператор SELECT COUNT(*) может прочитать некоторые, но не все вставленные записи другой транзакции, если фиксация этой транзакции произойдет во время чтения записей.

Снимок уровня оператора создается при выполнении оператора верхнего уровня, вложенные операторы (триггеры, хранимые процедуры и функции, операторы динамики и т. д.) используют снимок уровня оператора, созданный для оператора верхнего уровня.

Обработка конфликтов обновлений

Когда оператор выполняется в транзакции с режимом изоляции READ COMMITTED READ CONSISTENCY он видит неизменное состояние базы данных (подобно транзакции SNAPSHOT). Поэтому не имеет смысла ждать подтверждения параллельной транзакции в надежде перечитать новую версию подтверждённой записи. При чтении поведение похоже на транзакцию READ COMMITTED RECORD_VERSION — оператор не ждёт завершения активной транзакции и обходит цепочку версий, в которой ищет версию записи, видимую на текущем моментальном снимке.

Для уровня изоляции READ COMMITTED READ CONSISTENCY обработка конфликтов обновлений значительно изменяется. При обнаружении конфликта обновления выполняется следующее:

1. Режим изолированности транзакции временно переключается в режим READ COMMITTED NO RECORD_VERSION.
2. Устанавливается блокировка записи на конфликтную запись.
3. Сервер оценивает оставшиеся записи для удаления/обновления в курсоре, а также продолжает ставить на них блокировки.
4. Когда больше нет записей для извлечения, сервер начинает отменять все действия, выполненные с момента начала выполнения оператора верхнего уровня, и сохраняет все установленные блокировки для каждой обновлённой/удалённой/заблокированной записи, все вставленные записи удаляются.
5. Затем восстанавливается уровень изоляции транзакции READ COMMITTED READ CONSISTENCY, создается новый снимок уровня оператора и перезапускается выполнение оператора верхнего уровня.

Такой алгоритм гарантирует, что после перезапуска уже обновленные записи останутся заблокированными, они будут видны на новом снимке и могут быть обновлены снова без дальнейших конфликтов. Кроме того, из-за режима согласованности чтения набор измененных записей остается согласованным.

Замечания:

- Приведенный выше алгоритм перезапуска применяется к операторам UPDATE, DELETE, SELECT WITH LOCK и MERGE, с предложением RETURNING и без него, выполняемым непосредственно из пользовательского приложения или в составе некоторого объекта PSQL (хранимая процедура, функция, триггер, EXECUTE BLOCK и т. д.).

- Если оператор UPDATE/DELETE расположен на каком-либо явном курсоре (WHERE CURRENT OF), то пропускается шаг выше, то есть не извлекаются и не устанавливаются блокировки записи для оставшихся записей курсора.
- Если оператор верхнего уровня SELECT (или EXECUTE BLOCK возвращающий набор данных) и конфликт обновления происходит после того, как одна или несколько записей были возвращены приложению, то ошибка конфликта обновления сообщается как обычно и перезапуск не инициируется.
- Перезапуск не инициируется для операторов в автономных блоках (IN AUTONOMOUS TRANSACTION DO ...).
- После 10 попыток прерывается алгоритм перезапуска, снимаются все блокировки записи, восстанавливается режим изоляции транзакции как READ COMMITTED READ CONSISTENCY и сообщается о конфликте обновления.
- Любая необработанная ошибка на шаге выше останавливает алгоритм перезапуска и сервер продолжает работать в обычном режиме, например, ошибка может быть перехвачена и обработана PSQL-блоком WHEN или возвращена пользователю, если она не была обработана.
- Триггеры UPDATE/DELETE сработают многократно для одной и той же записи, если выполнение оператора было перезапущено и запись обновлена/удалена снова.
- По историческим причинам isc_update_conflict сообщается как вторичный код ошибки с основным кодом ошибки isc_deadlock.

5.1.6 Опция NO AUTO UNDO

При использовании опции NO AUTO UNDO оператор ROLLBACK только помечает транзакцию как отменённую без удаления созданных в этой транзакции версий, которые будут удалены позднее в соответствии с выбранной политикой сборки мусора (см. параметр GCPolicy в firebird.conf).

Эта опция может быть полезна при выполнении транзакции, в рамках которой производится много отдельных операторов, изменяющих данные, и при этом есть уверенность, что эта транзакция будет завершаться успешно, а не откатываться.

Для транзакций, в рамках которых не выполняется никаких изменений, опция NO AUTO UNDO игнорируется.

5.1.7 Опция AUTO COMMIT

Указание AUTO COMMIT включает режим автокоммита для транзакции. В режиме автоматической фиксации сервер будет внутренне выполнять эквивалент COMMIT RETAIN после каждого выполнения оператора.

Это не совсем удобный режим автокоммита; контекст транзакции сохраняется до тех пор, пока транзакция не будет завершена фиксацией или откатом. Другими словами, при использовании SNAPSHOT или SNAPSHOT TABLE STABILITY эта автокоммитация не изменит видимость записей (эффекты транзакций, которые были зафиксированы после запуска этой транзакции, не будут видны).

Для READ COMMITTED действуют те же предупреждения, что и для COMMIT RETAIN: длительное использование одной транзакции в режиме автокоммита может препятствовать сборке мусора и снижать производительность.

5.1.8 Опция IGNORE LIMBO

При указании опции IGNORE LIMBO игнорируются записи, создаваемые "потерянными" (т.е. не завершёнными) транзакциями (limbo transaction). Транзакции считается "потерянной", если не завершён второй этап двухфазного подтверждения (two-phase commit).

5.1.9 Опция AUTO RELEASE TEMP BLOBID

При использовании опции `AUTO RELEASE TEMP BLOBID` транзакция автоматически освобождает временный идентификатор пользовательского `BLOB` сразу после его материализации. Это полезно при массовых вставках записей с пользовательскими `BLOB`, так как при этом устраняются затраты памяти на хранение временных идентификаторов.

Опцию `AUTO RELEASE TEMP BLOBID` следует использовать с осторожностью и только в том случае, если нет необходимости обращаться к материализованному `BLOB` через временный идентификатор. РЕД База Данных автоматически применяет эту опцию при выполнении восстановления базы данных.

5.1.10 Предложение RESERVING

Предложение `RESERVING` в операторе `SET TRANSACTION` резервирует таблицы, указанные в списке таблиц.

Резервирование таблицы не позволяет другим транзакциям вносить в нее изменения или даже, при включении определенных параметров, считывать из нее данные во время выполнения этой транзакции.

Предложение `RESERVING` также может использоваться для указания списка таблиц, которые могут быть изменены другими транзакциями, даже если транзакция запущена с уровнем изоляции `SNAPSHOT TABLE STABILITY`.

Синтаксис предложения резервирования представлен в листинге 5.2.

Листинг 5.2. Синтаксис предложения резервирования

```
RESERVING <предложение резервирования>

<предложение резервирования> ::= <имя таблицы> [, <имя таблицы> ...]
                                 [FOR [SHARED | PROTECTED] {READ | WRITE}]
                                 [, <предложение резервирования>] ...
```

Если одно из ключевых слов `SHARED` или `PROTECTED` опущено, предполагается `SHARED`. Если опущен весь пункт `FOR`, предполагается `FOR SHARED READ`.

Названия и совместимость четырех вариантов доступа для резервирования таблиц:

Таблица 5.2 — Совместимости различных блокировок

	SHARED READ	SHARED WRITE	PROTECTED READ	PROTECTED WRITE
SHARED READ	да	да	да	да
SHARED WRITE	да	да	нет	нет
PROTECTED READ	да	нет	да	нет
PROTECTED WRITE	да	нет	нет	нет

Комбинации этих флагов пункта `RESERVING` для параллельного доступа зависят от уровней изоляции параллельных транзакций:

- `SNAPSHOT`

  Параллельные транзакции `SNAPSHOT` с `SHARED READ` не влияют на доступ друг друга.

  Параллельные транзакции `SNAPSHOT` и `READ COMMITTED` с `SHARED WRITE` не влияют на доступ друг друга, но блокируют транзакции с изоляцией `SNAPSHOT TABLE STABILITY` от чтения из или записи в указанную таблицу.

  Параллельные транзакции с любым уровнем изоляции и `PROTECTED READ` могут только читать данные из зарезервированных таблиц. Любая попытка записи в них вызовет исключение.

При использовании PROTECTED WRITE одновременные транзакции с изоляцией SNAPSHOT и READ COMMITTED не могут записывать в указанные таблицы. Транзакции с изоляцией SNAPSHOT TABLE STABILITY вообще не могут читать из зарезервированных таблиц или писать в них.

- SNAPSHOT TABLE STABILITY

  Все параллельные транзакции с SHARED READ, независимо от их уровней изоляции, могут читать из зарезервированных таблиц или записывать в них (если они находятся в режиме READ WRITE).

  Параллельные транзакции с уровнями изоляции SNAPSHOT и READ COMMITTED и SHARED WRITE могут читать данные из указанных таблиц и записывать их (если находятся в режиме READ WRITE), но одновременный доступ к этим таблицам со стороны транзакций с SNAPSHOT TABLE STABILITY блокируется, пока эти транзакции активны.

  Параллельные транзакции с любым уровнем изоляции и PROTECTED READ могут читать только из зарезервированных таблиц.

  С PROTECTED WRITE одновременные транзакции SNAPSHOT и READ COMMITTED могут читать из зарезервированных таблиц, но не записывать в них. Доступ транзакций с уровнем изоляции SNAPSHOT TABLE STABILITY блокируется.

- READ COMMITTED

  При SHARED READ все параллельные транзакции с любым уровнем изоляции могут как читать из зарезервированных таблиц, так и записывать в них (если они находятся в режиме READ WRITE).

  SHARED WRITE позволяет всем транзакциям с изоляцией SNAPSHOT и READ COMMITTED читать из указанных таблиц и записывать в них (если в режиме READ WRITE) и блокирует доступ транзакций с изоляцией SNAPSHOT TABLE STABILITY.

  При PROTECTED READ параллельные транзакции с любым уровнем изоляции могут читать только из зарезервированных таблиц.

  С PROTECTED WRITE параллельные транзакции в изоляции SNAPSHOT и READ COMMITTED могут читать из указанных таблиц, но не записывать в них. Доступ от транзакций в изоляции SNAPSHOT TABLE STABILITY блокируется.

5.2 Подтверждение транзакции

Чтобы подтвердить текущую транзакцию используется оператор COMMIT (см. листинг 5.3).

Листинг 5.3. Синтаксис оператора подтверждения транзакции COMMIT

```
COMMIT [WORK] [TRANSACTION <имя транзакции>]
[RELEASE] [RETAIN [SNAPSHOT]];
```

При выполнении этого оператора подтверждаются все изменения в данных, выполненные в контексте данной транзакции (добавления, изменения, удаления). Новые версии записей становятся доступными для других процессов. Если не указано предложение RETAIN, то при этом освобождаются все ресурсы сервера, связанные с выполнением данной транзакции. Если в процессе подтверждения транзакции возникли ошибки в базе данных, то транзакция не подтверждается. Пользовательская программа должна обработать ошибочную ситуацию и заново подтвердить транзакцию или выполнить ее откат.

Необязательное ключевое слово WORK может быть использовано лишь для совместимости с другими системами управления реляционными базами данных.

Необязательное предложение TRANSACTION задаёт имя транзакции. Предложение TRANSACTION доступно только в Embedded SQL. Если предложение TRANSACTION не указано, то оператор COMMIT применяется к транзакции по умолчанию.

Ключевое слово RELEASE доступно только в Embedded SQL. Оно позволяет отключиться ото всех

баз данных после завершения текущей транзакции. RELEASE поддерживается только для обратной совместимости со старыми версиями серверов базы данных.

Если используется предложение RETAIN [SNAPSHOT], то выполняется так называемое мягкое (soft) подтверждение. Выполненные действия в контексте данной транзакции фиксируются в базе данных, а сама транзакция продолжает оставаться активной. В этом случае нет необходимости опять стартовать транзакцию и заново выдавать оператор SELECT для получения данных из таблицы. Если уровень изоляции такой транзакции SNAPSHOT или SNAPSHOT TABLE STABILITY, то после мягкого подтверждения транзакция продолжает видеть то состояние базы данных, которое было при первоначальном запуске транзакции, то есть клиентская программа не видит новых подтвержденных результатов изменения данных других процессов. Кроме того, мягкое подтверждение не освобождает ресурсов сервера.

> Для транзакций, которые выполняют только чтение данных из базы данных, рекомендуется также использовать оператор COMMIT, а не ROLLBACK, поскольку этот вариант требует меньшего количества ресурсов сервера и улучшает производительность всех последующих транзакций.

5.3 Откат (отмена) транзакции

Для отмены всех изменений, выполненных в контексте текущей транзакции, или для отката на созданную ранее в контексте транзакции контрольную точку используется оператор ROLLBACK.

Листинг 5.4. Синтаксис оператора отката транзакции ROLLBACK

```
ROLLBACK  [WORK] [TRANSACTION <имя транзакции>]
[RETAIN [SNAPSHOT] | TO SAVEPOINT <имя точки сохранения>] [RELEASE];
```

При выполнении оператора отменяются все изменения данных базы данных (добавление, изменение, удаление), выполненные под управлением этой транзакции. Оператор ROLLBACK никогда не вызывает ошибок. Если не указано предложение RETAIN, то при его выполнении освобождаются все ресурсы сервера, связанные с выполнением данной транзакции.

Необязательное ключевое слово WORK может быть использовано лишь для совместимости с другими системами управления реляционными базами данных.

Необязательное предложение TRANSACTION задаёт имя транзакции. Предложение доступно только в Embedded SQL. Если предложение TRANSACTION не указано, то оператор ROLLBACK применяется к транзакции по умолчанию.

Необязательное предложение TO SAVEPOINT задает имя точки сохранения, на которую происходит откат. Подробнее о вложенных транзакциях и точках сохранения см. в следующем разделе.

Ключевое слово RETAIN указывает, что все действия по изменению данных в контексте этой транзакции, отменяются, при этом контекст транзакции сохраняется. Выделенные ресурсы для транзакции не освобождаются. Для уровней изоляции SNAPSHOT и SNAPSHOT TABLE STABILITY состояние базы данных остается в том виде, которое база данных имела при первоначальном старте такой транзакции, однако в случае уровня изоляции READ COMMITTED база данных будет иметь вид, соответствующий новому состоянию на момент выполнения оператора ROLLBACK RETAIN. В случае отмены транзакции с сохранением ее контекста нет необходимости заново выполнять оператор SELECT для получения данных из таблицы.

Ключевое слово RELEASE доступно только в Embedded SQL. Оно позволяет отключиться ото всех баз данных после завершения текущей транзакции. RELEASE поддерживается только для обратной совместимости со старыми версиями серверов базы данных.

5.4 Использование вложенных транзакций

Вложенные транзакции (**nested transactions**) позволяют в процессе выполнения действий с базой данных в контексте одной длинной транзакции создавать некоторые контрольные точки, или точки сохранения, к которым можно вернуться, не отменяя действий всей транзакции. В этом случае состояние базы данных станет соответствовать тому состоянию, которое база данных имела на момент создания этой точки сохранения (здесь учитываются только те изменения, которые были выполнены в контексте данной транзакции). В процессе активности транзакции можно создавать произвольное количество точек сохранения. Они упорядочиваются в хронологическом порядке — по мере их создания.

Для создания точки сохранения используется оператор SAVEPOINT. Его синтаксис представлен в листинге 5.5.

Листинг 5.5. Синтаксис оператора создания точки сохранения SAVEPOINT

```
SAVEPOINT <имя точки сохранения>;
```

Имя точки сохранения — обычный идентификатор базы данных, который может содержать до 63 символов.

Имена точек сохранения, созданных в контексте одной транзакции, должны отличаться. Если же в операторе SAVEPOINT создается точка сохранения с именем, уже присутствующем в списке созданных точек сохранения в процессе активности данной транзакции, то существующая точка сохранения будет удалена, и создаётся новая с тем же именем.

Любую созданную в транзакции точку сохранения можно удалить, выполнив оператор RELEASE SAVEPOINT. Синтаксис оператора представлен в листинге 5.6.

Листинг 5.6. Синтаксис оператора удаления точки сохранения RELEASE SAVEPOINT

```
RELEASE SAVEPOINT <имя точки сохранения> [ONLY];
```

Оператор удаляет указанную по ее имени точку сохранения из списка. Если не указано ключевое слово ONLY, то удаляются и все последующие точки сохранения. Ключевое слово ONLY задает удаление только одной указанной точки сохранения без какого-либо влияния на другие последующие.

Если точка сохранения с таким именем отсутствует, то не выдается никакого сообщения об ошибке. Операция удаления точки сохранения «молчаливо» не выполняется.

Для отката транзакции на одну из созданных ранее в контексте этой транзакции точек сохранения используется оператор ROLLBACK TO SAVEPOINT. Его синтаксис в этом случае выглядит следующим образом:

Листинг 5.7. Синтаксис оператора отката транзакции на точку сохранения ROLLBACK TO SAVEPOINT

```
ROLLBACK TO SAVEPOINT <имя точки сохранения>
[WORK];
```

Ключевое слово WORK используется только лишь для совместимости с другими реляционными системами управления базами данных и со стандартом SQL-92.

Оператор отменяет только те изменения, которые были сделаны в базе данных в контексте данной транзакции после создания указанной точки сохранения.

Оператор ROLLBACK TO SAVEPOINT выполняет следующие операции:

- Все изменения в базе данных, выполненные в рамках транзакции начиная с созданной точки сохранения, отменяются. Пользовательские переменные, заданные с помощью функции RDB$SET_CONTEXT() остаются неизменными.

- Все точки сохранения, создаваемые после названной, уничтожаются. Все более ранние точки сохранения, как сама точка сохранения, остаются. Это означает, что можно откатываться к той же точке сохранения несколько раз.
- Все явные и неявные блокированные записи, начиная с точки сохранения, освобождаются. Другие транзакции, запросившие ранее доступ к строкам, заблокированным после точки сохранения, должны продолжать ожидать, пока транзакция не фиксируется или откатывается. Другие транзакции, которые ещё не запрашивали доступ к этим строкам, могут запросить и сразу же получить доступ к разблокированным строкам.

Транзакция продолжает оставаться активной, как если бы было задано ключевое слово `RETAIN`.

Если точка сохранения с таким именем отсутствует, то не выдается никакого сообщения. Операция отката просто не выполняется.

5.5 Внутренние точки сохранения

По умолчанию сервер использует автоматическую системную точку сохранения уровня транзакции для выполнения её отката. При выполнении оператора `ROLLBACK`, все изменения, выполненные в транзакции, откатываются до системной точки сохранения и после этого транзакция подтверждается.

Когда объем изменений, выполняемых под системной точкой сохранения уровня транзакции, становится большим (затрагивается порядка 50000 записей) сервер освобождает системную точку сохранения и, при необходимости отката транзакции, использует механизм TIP.

5.6 Точки сохранения и PSQL

Использование операторов управления транзакциями в PSQL не разрешается, так как это нарушит атомарность оператора, вызывающего процедуру. Но РЕД База Данных поддерживает вызов и обработку исключений в PSQL, так, чтобы действия, выполняемые в хранимых процедурах и триггерах, могли быть выборочно отменены без полного отката всех действий в них. Внутренне автоматические точки сохранения используется для:

- отмены всех действий внутри блока `BEGIN ... END`, где происходит исключение;
- отмены всех действий, выполняемых в хранимой процедуре/триггере (или, в случае селективной хранимой процедуры, всех действий, выполненных с момента последнего оператора `SUSPEND`), если они завершаются преждевременно из-за непредусмотренной ошибки или исключения.

Каждый блок обработки исключений PSQL также ограничен автоматическими точками сохранения сервера.

Сам по себе блок `BEGIN ... END` не создаёт автоматическую точку сохранения. Она создаётся только в блоках, которых присутствует блок `WHEN` для обработки исключений или ошибок.

5.7 Вариант взаимной блокировки

Существует вероятность того, что две конкурирующие транзакции создадут ситуацию взаимной блокировки, или как ее еще называют «смертельная блокировка» (`dead lock`). Такая взаимная блокировка произойдет, если первая транзакция ожидает завершения второй транзакции (подтверждения или отмены), а вторая транзакция ожидает в том или ином виде завершения первой транзакции. Для появления взаимной блокировки обе конкурирующие транзакции должны иметь режим разрешения блокировки `WAIT`. Их уровни изоляции могут быть `SNAPSHOT` или `READ COMMITTED`. Взаимная блокировка возможна и в случае уровня изоляции транзакции `SNAPSHOT TABLE STABILITY`, если предложение резервирования при старте этой транзакции дает возможность другим транзакциям изменять данные отдельных таблиц базы данных.

Например, первая транзакция изменила в некоторой таблице данные первой строки и не под-

твердила изменения. Вторая транзакция изменила данные второй строки той же таблицы и также не подтвердила эти изменения. После этого первая транзакция пытается изменить вторую строку. Поскольку в другой транзакции выполнены неподтвержденные изменения этой строки, первая транзакция переходит в режим ожидания. Далее вторая транзакция, являясь активной и дееспособной, пытается изменить первую строку таблицы, неподтвержденные изменения которой выполнила первая транзакция. Вторая транзакция тут же перейдет в режим ожидания. Обе транзакции будут ожидать соответствующих действий друг от друга, в этом случае создается взаимная блокировка.

В РЕД База Данных существует Менеджер блокировок (`Lock Manager`), который отслеживает и обрабатывает подобные ситуации взаимных блокировок. Менеджер блокировок вызывается с определенной периодичностью, которая задается в файле конфигурации системы `firebird.conf` параметром `DeadlockTimeout`. Значением по умолчанию является интервал в 10 секунд. По истечении этого срока Менеджер блокировок разрешит ситуацию взаимной блокировки, создав исключительную ситуацию («взаимная блокировка, изменение конфликтует с параллельным изменением») для одной из транзакций, включенных в конфликтную ситуацию.

Глава 6

Базы данных (DATABASE)

Прежде, чем начать создавать объекты базы данных и заполнять базу данными конкретной предметной области, необходимо создать базу данных с необходимыми характеристиками.

6.1 Создание базы данных

Для создания базы данных используется оператор SQL CREATE DATABASE. Его синтаксис в нотациях Бэкуса-Наура представлен в листинге 6.1.

Листинг 6.1. Синтаксис оператора создания базы данных CREATE DATABASE

```
CREATE {DATABASE | SCHEMA} '<спецификация файла>'
   [USER '<имя пользователя>' [PASSWORD '<пароль>'] [ROLE <имя роли>]]
   [PAGE_SIZE [=] <целое>]
   [LENGTH [=] <целое> [PAGE[S]]]
   [SET NAMES '<набор символов>']
   [DEFAULT CHARACTER SET <набор символов>
      [COLLATION <сортировка по умолчанию>]]
   [DIFFERENCE FILE '<имя файла>']
   [<вторичный файл> [<вторичный файл>...]]

<спецификация файла> ::=
            [<спецификация удалённого сервера>] { <путь к файлу БД> | <алиас БД> }

<спецификация удалённого сервера> ::=
              <хост>[\<порт> | <имя сервиса>]:
            | \\<хост>[@<порт> | <имя сервиса>]\
            | <протокол>://[ <имя сервиса>[: <порт> | <имя сервиса>]/]

<протокол> = inet | inet4 | inet6 | xnet

<вторичный файл> ::= FILE '<путь к файлу>'
                     [LENGTH [=] <целое> [PAGE[S]]]
                     [STARTING [AT [PAGE]] <целое>]
```

Можно использовать оператор CREATE DATABASE или CREATE SCHEMA. Это синонимы.

Создать новую базу данных может только администратор и пользователь с привилегией CREATE DATABASE.

База данных может состоять из одного или нескольких файлов. Первый, основной, файл называется первичным, остальные файлы — вторичными. В настоящее время многофайловые базы данных являются атавизмом. Многофайловые базы данных имеет смысл использовать на старых файловых системах, в которых существует ограничение на размер любого файла. Например, в FAT32 нельзя создать файл больше 4х гигабайт.

Для того, чтобы база данных была создана в нужном вам диалекте SQL, следует перед выполнением оператора создания базы данных задать нужный диалект, выполнив оператор SET SQL DIALECT:

```
SET SQL DIALECT 3;
```

Для вновь создаваемых баз данных по умолчанию используется диалект 3, который позволяет более эффективно использовать внешнюю память и средства системы управления базами данных РЕД Базы Данных.

6.1.1 Спецификация файла

Спецификация файла — имя файла базы данных и его расширение с указанием к нему полного пути в соответствии с правилами используемой операционной системы. Сам файл должен отсутствовать на диске. В противном случае будет выдано диагностическое сообщение, и база данных не будет создана. Если полный путь к базе данных не указан, то база данных будет создана в одном из системных каталогов. В каком именно зависит от операционной системы.

Файл может иметь любое расширение или не иметь вообще никакого расширения. Для файлов СУБД РЕД База Данных принято использовать расширение `.fdb`.

Вместо полного пути к первичному файлу базы можно использовать псевдонимы (aliases). Псевдонимы описываются в файле `databases.conf` в формате:

```
<алиас БД> = <путь к файлу БД>
```

6.1.2 Спецификация удаленного сервера

При создании базы данных на удалённом сервере необходимо указать спецификацию удалённого сервера. Спецификация удалённого сервера зависит от используемого протокола.

- Если вы при создании базы данных используете протокол TCP/IP, то спецификация первичного файла должна выглядеть следующим образом:

  ```
  <хост>[\<порт> | <имя сервиса>]:{ <путь к файлу БД> | <алиас БД> }
  ```

  Например, следующая спецификация файла использует этот протокол для размещения вновь создаваемого файла базы данных на сервере с именем в сети Server, на диске D в каталоге RedSoftDatabase:

  ```
  'Server:D:\RedSoftDatabase\work.fdb'
  ```

- Если вы при создании базы данных используете протокол под названием именованные каналы (Name Pipes), то спецификация первичного файла должна выглядеть следующим образом:

  ```
  \\<хост>[@<порт> | <имя сервиса>]\{ <путь к файлу БД> | <алиас БД> }
  ```

  Аналогичный путь к файлу базы данных можно задать следующим образом:

  ```
  '\\Server\D:\RedSoftDatabase\work.fdb'
  ```

- Существует также унифицированный URL-подобный синтаксис спецификации удалённого сервера. В этом синтаксисе первым параметром указывается наименование протокола, далее указывается имя сервера или IP адрес, номер порта и путь к первичному файлу базы данных или псевдоним. В качестве протокола можно указать следующие значения:

INET	TCP/IP (сначала пробует подключится по протоколу TCP/IP v6, если не получилось, то TCP/IP v4)
INET4	TCP/IP v4
INET6	TCP/IP v6
XNET	Локальный протокол

```
<протокол>://[ <имя сервиса>[:<порт>|<имя сервиса>]/]{ <путь БД>|<алиас БД>}
```

6.1.3 USER, PASSWORD, ROLE

Необязательные предложения в операторе создания базы данных `USER` и `PASSWORD` задают, соответственно, имя и пароль пользователя, присутствующего в базе данных безопасности `security5.fdb`. Файл базы данных безопасности находится в корневом каталоге инсталляции РЕД Базы Данных. Заданный в предложении `USER` пользователь становится владельцем созданной базы данных и имеет к ней неограниченные полномочия.

Пользователя и пароль можно не указывать, если установлены переменные окружения `ISC_USER` и `ISC_PASSWORD`. Имя пользователя и его пароль будут выбраны из этих переменных окружения. Использование таких переменных окружения не рекомендуется в промышленно работающих системах, так как подобная практика резко ухудшает безопасность системы. Подробнее см. документ «Руководство администратора».

Необязательное предложение `ROLE` задаёт имя роли (обычно это `RDB$ADMIN`), права которой будут учитываться при создании базы данных. Роль должна быть назначена пользователю в соответствующей базе данных безопасности.

После инсталляции РЕД Базы Данных на компьютере база данных безопасности содержит ровно одного пользователя `SYSDBA` с паролем `masterkey`. Это особый пользователь, администратор всех баз данных, расположенных на сервере. Этот пользователь может создавать учетные записи других пользователей и имеет неограниченные полномочия к любой базе данных, располагающейся на данном компьютере.

6.1.4 PAGE_SIZE

Необязательное предложение `PAGE_SIZE` задает размер страницы базы данных в байтах. Этот размер страницы будет установлен как для первичного, так и для всех вторичных файлов создаваемой базы данных в том случае, если создается многофайловая база. Допустимыми значениями являются 4096, 8192 (значение по умолчанию), 16384 или 32768. Если вы зададите неправильное значение размера страницы, то система установит размер до ближайшего меньшего числа. Если указать значение меньше чем 4096, то будет выбрано значение — 4096.

6.1.5 LENGTH

Предложение `LENGTH` задает максимальный размер первичного или вторичного файла базы данных в страницах. При создании базы данных любой файл (первичный или вторичный) независимо от значения `LENGTH` будет иметь минимально необходимый размер как минимум для хранения системных данных. Для единственного или последнего файла базы данных значение, указанное в предложении `LENGTH`, никак не влияет на величину используемой файлом памяти. Размер файла будет при необходимости автоматически увеличиваться в процессе добавления новых строк в таблицы до максимальной величины, которую обеспечивает используемая операционная система, или пока не будет исчерпано дисковое пространство носителя, на котором располагается этот файл.

6.1.6 SET NAMES, DEFAULT CHARACTER SET

Необязательное предложение `SET NAMES` задаёт набор символов подключения, доступного после успешного создания базы данных. По умолчанию используется набор символов `NONE`.

Предложение `DEFAULT CHARACTER SET` задает набор символов по умолчанию для строковых (символьных) данных для всей базы данных. Наборы символов применяются только для типов данных `CHAR, VARCHAR` и `BLOB`. Если для символьного столбца не указать набора символов, то ему будет присвоен набор символов `NONE`, иными словами, никакой. Работа с данными такого столбца крайне затруднительна. Помимо набора символов строковым столбцам задаются и соответствующие указанным наборам символов допустимые порядки сортировки. Для столбцов, которые будут содержать помимо латинских букв и спецсимволов также и буквы кириллицы, следует задавать набор символов `WIN1251`, а в качестве порядка сортировки желательно использовать `PXW_CYRL`, чтобы сортировка данных выполнялась в соответствии с правилами русского языка. Списки наборов символов и допустимых порядков сортировки для каждого набора символов представлены в Приложение Д.

Необязательный параметр `COLLATION`, связанный с набором символов, позволяет создавать все текстовые столбцы, домены и переменные с указанной последовательностью сортировки, если не указан другой `COLLATE`.

Сортировка по умолчанию для набора символов может быть изменена с помощью `ALTER CHARACTER SET`.

6.1.7 DIFFERENCE FILE

Ключевое слово `DIFFERENCE FILE` задаёт путь и имя дельта файла, в который будут записываться изменения, внесённые в БД после перевода её в режим «безопасного копирования» («copy-safe») путём выполнения команды `ALTER DATABASE BEGIN BACKUP`.

6.1.8 Вторичные файлы

База данных может состоять более чем из одного файла. Первый, основной, файл называется первичным, остальные — вторичными. Количество вторичных файлов произвольно, оно ограничивается только возможностями используемой операционной системы. Файлы базы данных могут располагаться на различных носителях серверной машины. Для них обычно используются расширения `.fd2, .fd3` и т.д.

При создании многофайловой базы данных необходимо либо для предыдущего файла в списке указать предложение `LENGTH`, либо для каждого из последующих файлов задавать предложение `STARTING AT`. В одном операторе создания базы данных могут одновременно использоваться и предложения `LENGTH`, и предложения `STARTING AT` (см. раздел 6.2 Примеры создания базы данных).

Предложение `STARTING AT` задает номер страницы базы данных, с которой должен начинаться следующий вторичный файл. Когда предыдущий файл будет полностью заполнен данными в соответствии с заданным номером страницы следующего файла, система начнет помещать вновь добавляемые данные в следующий вторичный файл. Управлять размещением в разные файлы отдельных добавляемых новых строк в таблицы базы данных пользователь не имеет возможности.

Если для первичного файла указать слишком малое количество страниц, в которые не смогут поместиться все системные данные, размещаемые при первоначальном создании базы данных (системные таблицы, ссылки на вторичные файлы, на оперативные копии и др.), то система все равно распределит для первичного файла соответствующее количество страниц, необходимое для хранения системных данных.

Средствами операторов SQL можно лишь добавить новые вторичные файлы к существующей базе данных — см. оператор `ALTER DATABASE`. Другие характеристики изменить таким образом невоз-

можно. Размер страницы, количество и размеры вторичных файлов можно изменять, выполняя резервное копирование и восстановление базы данных. Подробности см. в документе «Руководство администратора».

6.2 Примеры создания базы данных

Пример 1. Для создания однофайловой базы данных в `isql` или в любой соответствующей программе графического интерфейса нужно ввести и выполнить следующие операторы (имеется в виду операционная система Windows, в UNIX-подобных системах нужно только внести некоторые изменения в путь к файлу базы данных):

```
SET SQL DIALECT 3;
CREATE DATABASE 'localhost:D:\RedSoftDatabase\work.fdb'
   USER 'wizard' PASSWORD 'master'
   PAGE_SIZE = 32768
   DEFAULT CHARACTER SET WIN1251;
```

Здесь будет создана база данных в диалекте 3, владельцем которой является описанный в системе пользователь `wizard` с паролем `master` (этот пользователь должен быть создан в базе данных безопасности до создания демонстрационной базы данных — см. документ «Руководство администратора»). Размер страницы для этой базы данных установлен максимальным — 32768.

Мы предполагаем, что в строковых типах данных таблиц будут присутствовать и буквы кириллицы, поэтому указываем набор символов по умолчанию для строковых данных базы данных `WIN1251`. Для отдельных столбцов таблиц можно задать наборы символов, отличные от набора символов по умолчанию — см. главу 9.

В `isql` можно отобразить состояние созданной базы данных, соединившись с базой данных (для этого используется оператор `CONNECT` — см. далее в этой главе раздел 6.3) и выполнив оператор `isql SHOW DATABASE`:

```
SHOW DATABASE;

Owner: WIZARD
PAGE_SIZE 32768
Number of DB pages allocated = 199
Number of DB pages used = 184
Number of DB pages free = 15
Sweep interval = 20000
Forced Writes are ON
Transaction - oldest = 6
Transaction - oldest active = 7
Transaction - oldest snapshot = 7
Transaction - Next = 10
ODS = 13.1
Database not encrypted
Wire crypt plugin: ChaCha64
Creation date: May 30, 2024 15:42:36
Replica mode: NONE
Protocol version = 18
Default Character set: WIN1251
Publication: Disabled
```

Некоторые характеристики созданной базы данных можно просмотреть и в различных программах графического интерфейса, предназначенных для работы с базами данных.

Пример 2. Пусть демонстрационная база будет использовать три файла — один первичный и два вторичных. Тогда в предыдущие операторы следует добавить некоторые изменения, задав пути к вторичным файлам и значения номеров страниц, с которых должны начинаться вторичные файлы:

```
SET SQL DIALECT 3;
CREATE DATABASE 'localhost:D:\RedSoftDatabase\work.fdb'
   USER 'wizard' PASSWORD 'master'
   PAGE_SIZE = 4096
   DEFAULT CHARACTER SET WIN1251
   FILE 'D:\RedSoftDatabase\work.fd2'
      STARTING AT PAGE 10001
   FILE 'D:\RedSoftDatabase\work.fd3'
      STARTING AT PAGE 20001;
```

Первичный файл будет содержать 10000 страниц размером 4096 байтов. Как только первичный файл в процессе работы с базой данных будет заполнен пользовательскими и системными данными (количество всех данных превысит 4096×10000 байтов, включая и системные данные — заголовки страниц, указатели, индексы, генераторы и т.д.), система управления базами данных начнет помещать новые строки таблиц во второй файл базы данных, `work.fd2`. Аналогичные действия произойдут, когда будет заполнен и этот вторичный файл. Система начнет помещать данные в следующий вторичный файл — `work.fd3`. Размер последнего вторичного файла будет увеличиваться до того предела, который допускает используемая версия операционной системы, или пока не будет исчерпана память на внешнем носителе. Управлять размещением в различных файлах базы данных отдельных строк для различных таблиц пользователь не имеет возможности, все эти действия выполняет система управления базами данных.

Выполнив оператор `SHOW DATABASE`, можно увидеть результат создания такой базы данных:

```
SHOW DATABASE;

Owner: WIZARD
File 1: 'localhost:D:\RedSoftDatabase\work.fdb', length 10000, start 10001
File 2: 'D:\RedSoftDatabase\work.fd2', length 0, start 20001
PAGE_SIZE 4096
Number of DB pages allocated = 199
Number of DB pages used = 184
Number of DB pages free = 15
Sweep interval = 20000
Forced Writes are ON
Transaction - oldest = 6
Transaction - oldest active = 7
Transaction - oldest snapshot = 7
Transaction - Next = 10
ODS = 13.1
Database not encrypted
Wire crypt plugin: ChaCha64
Creation date: May 30, 2024 15:42:36
Replica mode: NONE
Protocol version = 18
Default Character set: WIN1251
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
Publication: Disabled
```

Пример 3. Точно такую же базу данных мы получим, если зададим оператор следующим образом:

```
SET SQL DIALECT 3;
CREATE DATABASE 'localhost:D:\RedSoftDatabase\work.fdb'
   USER 'wizard' PASSWORD 'master'
   PAGE_SIZE = 4096
   LENGTH = 10000 PAGES
   DEFAULT CHARACTER SET WIN1251
   FILE 'D:\RedSoftDatabase\work.fd2'
   LENGTH = 10000 PAGES
   FILE 'D:\RedSoftDatabase\work.fd3';
```

Здесь вместо предложения `STARTING AT` для вторичных файлов было использовано предложение `LENGTH` для первичного и первого вторичного файла.

Отобразив базу данных в `isql`, получаем такие же характеристики, что и в предыдущем примере.

Пример 4. Смешанный вариант задания предложений `LENGTH` и `STARTING AT`. Такую же многофайловую базу данных мы получим, если зададим оператор следующим образом:

```
SET SQL DIALECT 3;
CREATE DATABASE 'localhost:D:\RedSoftDatabase\work.fdb'
   USER 'wizard' PASSWORD 'master'
   PAGE_SIZE = 4096
   DEFAULT CHARACTER SET WIN1251
   FILE 'D:\RedSoftDatabase\work.fd2'
   LENGTH = 10000 PAGES STARTING AT PAGE 10001
   FILE 'D:\RedSoftDatabase\work.fd3';
```

Здесь в первом из вторичных файлов указано и предложение `STARTING AT`, и предложение `LENGTH`. Во втором вторичном файле в данном случае не задается предложение `STARTING AT`.

Отобразив базу данных в `isql`, получаем те же характеристики, что и в предыдущих двух примерах.

Пример 5. Наконец, можно и иначе перемешать предложения `STARTING AT` и `LENGTH`. В следующем примере для первичного файла указывается предложение `LENGTH`, для первого вторичного файла нет никаких соответствующих указаний, а для второго вторичного файла задается предложение `STARTING AT`. В результате будет создана точно такая же база данных, как и в предыдущих трех примерах:

```
SET SQL DIALECT 3;
CREATE DATABASE 'localhost:D:\RedSoftDatabase\work.fdb'
   USER 'wizard' PASSWORD 'master'
   PAGE_SIZE = 4096
   LENGTH = 10000 PAGES
   DEFAULT CHARACTER SET WIN1251
   FILE 'D:\RedSoftDatabase\work.fd2'
   FILE 'D:\RedSoftDatabase\work.fd3'
       STARTING AT PAGE 20001;
```

6.3 Соединение с существующей базой данных

После создания любой базы данных для работы с ней (для выполнения добавления, изменения, удаления метаданных, добавления, изменения, удаления, поиска данных) с этой базой данных вначале нужно соединиться. Для этого используется оператор CONNECT (доступно в ISQL). Его синтаксис представлен в листинге 6.2.

Листинг 6.2. Синтаксис оператора соединения с существующей базой данных CONNECT

```
CONNECT '<спецификация файла>'
   [USER '<имя пользователя>' [PASSWORD '<пароль>']]
   [CACHE <целое> [BUFFERS]]
   [ROLE '<имя роли>'];
```

В операторе соединения с базой данных указывается имя только первичного файла базы данных, независимо от того, существуют ли у этой базы данных вторичные файлы. Если на компьютере не заданы переменные окружения ISC_USER и ISC_PASSWORD, то обязательно нужно указать имя пользователя (предложение USER) и его пароль (PASSWORD). Имя пользователя и пароль можно не указывать, если пользователь операционной системы имеет статус **trusted user** — см. документ «Руководство администратора».

Предложение CACHE задает количество буферов кэш-памяти для соединения, чтобы указать для подключаемой программы количество сохраняемых в памяти доступных страниц базы данных. По умолчанию это значение равняется 256. Максимальное количество зависит от используемой операционной системы и доступной оперативной памяти. Во многих случаях лучше это значение оставить значением по умолчанию. Тогда операционная система, скорее всего, примет не худшее решение по размеру кэша.

Предложение ROLE задает имя роли, с которой пользователь соединяется с данной базой данных. Имя роли может содержать до 63 символов. Подробности создания и использования ролей см. в «Руководство администратора».

В настоящей версии РЕД База Данных с каждой базой данных на сервере может соединиться любой пользователь, описанный в системе.

После успешного соединения с базой данных пользователь может выполнять с ней необходимые действия. Каждое новое соединение с базой данных при работе с утилитой isql или программой графического интерфейса вызывает отключение от базы данных, с которой перед этим было выполнено соединение.

Перед соединением с базой данных необходимо установить диалект клиента при помощи оператора SET SQL DIALECT и набор символов клиента для строковых данных, используя оператор SET NAMES. Во избежание сложностей в использовании базы данных следует задавать тот же диалект, который был указан при создании базы данных.

При работе с базой данных из программы графического интерфейса имеет смысл в операторе SET NAMES задать тот же набор символов, который является набором символов по умолчанию для базы данных (предложение DEFAULT CHARACTER SET в операторе создания базы данных). Однако если работа с базой данных осуществляется из среды DOS при помощи утилиты isql и если в таблицах базы данных присутствуют буквы кириллицы, то для корректного отображения данных следует задать набор символов DOS866. Допустимые в системе наборы символов и порядки сортировки представлены в Приложение Д.

Синтаксис оператора SET SQL DIALECT (листинг 6.3):

Листинг 6.3. Синтаксис оператора задания диалекта клиента SET SQL DIALECT

```
SET SQL DIALECT {1 | 3};
```

Синтаксис оператора `SET NAMES` см. в листинге 6.4:

Листинг 6.4. Синтаксис оператора задания набора символов для клиента `SET NAMES`

```
SET NAMES <набор символов>;
```

Пример. Для соединения с любой из баз данных, созданных в примерах предыдущего раздела, при работе с любой программой графического интерфейса нужно использовать следующие операторы:

```
SET SQL DIALECT 3;
SET NAMES WIN1251;
CONNECT 'localhost:D:\RedSoftDatabase\work.fdb'
   USER 'wizard' PASSWORD 'master';
```

При работе из командной строки DOS оператор `SET NAMES` должен быть записан в следующем виде:

```
SET NAMES DOS866;
```

6.4 Строки подключения

Если Вы хотите подключиться к базе данных или создать ее, Вы должны, среди прочего, предоставить клиентскому приложению (или, если Вы программист, подпрограммам, которые Вы вызываете) строку подключения к базе данных. Строка соединения однозначно идентифицирует местоположение базы данных на Вашем компьютере, в локальной сети или даже в интернете.

6.4.1 Строка подключения к локальной базе данных

Строка локального подключения состоит из пути к базе данных и имени файла в формате файловой системы, используемой на серверной машине.

- На Linux и других Unix серверах, например:

```
/opt/RedDatabase/examples/empbuild/employee.fdb
```

- На Windows серверах, например:

```
C:\Biology\Data\Primates\Apes\populations.fdb
```

Многие клиенты пользуются относительными путями для подключения (например, `..\examples\empbuild\employee.fdb`). Но пользоваться ими нужно с осторожностью, т.к. не всегда очевидно как они будут расширены. Случайно можно подключиться к другой базе данных и изменения в них могут привести к катастрофическим последствиям.

Вместо пути к файлу лучше указывать алиас базы данных из файла `databases.conf`.

Получив строку локального подключения **без указания сетевого протокола**, клиент РЕД Базы Данных сначала попытается сделать прямое `embedded` соединение с файлом базы данных, **минуя аутентификацию**, но учитывая привилегии и ограничения для предоставленного имени пользователя и/или роли (если провайдер `Engine12` включен в `firebird.conf` или `databases.conf` параметр `Providers`). Если файл базы данных существует, но соединение не устанавливается, поскольку клиентский процесс не имеет необходимых прав доступа к файлу, выполняется попытка подключения клиент-сервер (провайдером `Loopback`) в следующем порядке:

1. используя TCP/IP протокол через `localhost`;
2. на Windows: используя XNET протокол.

Можно явно указывать определенный протокол в строке подключения (в стиле URL) и таким образом обойти попытку `embedded` подключения:

- `inet://zappa` (TCP/IP подключение, используя алиас на локальном компьютере)
- `inet:///opt/RedDatabase/examples/citylife.fdb` (TCP/IP подключение, используя абсолютный путь на локальном Posix компьютере)
- `inet://C:\Work\Databases\Drills.fdb` (TCP/IP подключение, используя абсолютный путь на локальном Windows компьютере)
- `xnet://security.db` (XNET подключение, используя алиас на локальном Windows компьютере)
- `xnet://C:\Programmas\Firebird\Firebird_3_0\security5.fdb` (XNET подключение, используя абсолютный путь на локальном Windows компьютере)

6.4.2 Строка подключение через TCP/IP

Если Вы при подключении используете протокол TCP/IP, то спецификация файла базы данных должна выглядеть следующим образом:

```
{<имя сервера>|<IP адрес>}[/<номер порта>|/<имя сервиса>]:{<абс. путь>|<алиас>}
```

Примеры:

- Для Linux/Unix:

```
pongo:/opt/RedDatabase/examples/empbuild/employee.fdb
bongo/3052:fury
112.179.0.1:/var/Firebird/databases/butterflies.fdb
localhost:blackjack.fdb
```

- Для Windows:

```
siamang:C:\Biology\Data\Primates\Apes\populations.fdb
sofa:D:\Misc\Friends\Rich\Lenders.fdb
inca/fb_db:D:\Traffic\Roads.fdb
127.0.0.1:Borrowers
```

6.4.3 URL-подобная строка подключения

Существует также унифицированный URL-подобный синтаксис спецификации удалённого сервера. В этом синтаксисе первым параметром указывается наименование протокола, далее указывается имя сервера или IP адрес, номер порта и путь к первичному файлу базы данных или псевдоним.

```
<протокол>://[{<имя сервера>|<IP адрес>}[:<номер порта>|:<имя сервиса>]/]{<абс. путь>|
<алиас>}
```

В качестве протокола можно указать следующие значения:

- INET — TCP/IP;
- XNET — локальный протокол.

Примеры:

- Для Linux/Unix:

```
inet://pongo//opt/RedDatabase/examples/empbuild/employee.fdb
inet://bongo:3052/fury
inet://112.179.0.1//var/Firebird/databases/butterflies.fdb
inet://localhost/blackjack.fdb
```

- Для Windows:

```
inet://siamang/C:\\Biology\Data\Primates\Apes\populations.fdb
inet://sofa:4044/D:\Misc\Friends\Rich\Lenders.fdb
```

6.5 Изменение существующей базы данных

В тех случаях, когда созданную и заполненную данными базу данных нужно изменить, добавив к существующему файлу (существующим файлам) дополнительные вторичные файлы, следует использовать оператор `ALTER DATABASE / ALTER SCHEMA`. Этот оператор можно выполнять только после успешного соединения с базой данных.

Синтаксис оператора представлен в листинге 6.5:

Листинг 6.5. Синтаксис оператора изменения базы данных `ALTER DATABASE`

```
ALTER {DATABASE | SCHEMA}
[ADD FILE <вторичный файл> [ADD FILE <вторичный файл> ...]]
[ADD DIFFERENCE FILE '<имя файла>' | DROP DIFFERENCE FILE]
[{BEGIN | END} BACKUP]
[SET DEFAULT CHARACTER SET <набор символов>]
[SET LINGER TO <секунды> | DROP LINGER]
[SET DEFAULT SQL SECURITY {DEFINER | INVOKER}]
[ENCRYPT WITH <плагин шифрования> [KEY <ключ шифрования>] | DECRYPT]
```

Изменять базу данных может ее владелец, пользователь с административными привилегиями и пользователь с привилегией `ALTER DATABASE`.

Оператор `ALTER DATABASE` изменяет структуру файлов базы данных или переключает её в состояние «безопасное для копирования»

Оператор `ALTER DATABASE` позволяет лишь добавлять к существующей базе данных дополнительные вторичные файлы. Изменить размер страницы или размеры первичного или вторичных файлов этим оператором невозможно. Подобные изменения можно выполнить путем копирования и последующего восстановления существующей базы данных. Подробности см. в документе «Руководство администратора».

6.5.1 Добавление вторичного файла

Предложение `ADD FILE` добавляет к существующей базе данных дополнительные вторичные файлы. Может быть указано произвольное количество добавляемых вторичных файлов. Описание вторичного файла в этом операторе аналогично тому, что представлено в описании оператора создания базы данных (см. раздел 6.1 этой главы).

Пример. Как только в предыдущем первичном или вторичных файлах будет заполнено 30000 страниц, СУБД будет помещать данные во вторичный файл `test4.fdb`.

```
ALTER DATABASE
ADD FILE 'D:\RedSoftDatabase\work.fd4'
STARTING AT PAGE 30001;
```

6.5.2 Изменение пути и имени дельта файла

Предложение `ADD DIFFERENCE FILE` задает путь и имя дельта файла, в который будут записываться изменения, внесенные в базу данных после перевода ее в режим «безопасного копирования» («copy-safe»). Этот оператор в действительности не добавляет файла. Он просто переопределяет умалчиваемые имя и путь файла дельты.

Для изменения существующих установок необходимо сначала удалить ранее указанное описание файла дельты с помощью оператора `DROP DIFFERENCE FILE`, а затем задать новое описание файла дельты. Если не переопределять путь и имя файла дельты, то он будет иметь тот же путь и имя, что и БД, но с расширением `.delta`.

Предложение `DROP DIFFERENCE FILE` удаляет описание (путь и имя) файла дельты. На самом деле файл не удаляется. Он удаляет путь и имя файла дельты и при последующем переводе БД в режим «безопасного копирования» будут использованы значения по умолчанию (т.е. тот же путь и имя что и у файла БД, но с расширением `.delta`).

6.5.3 Перевод базы данных в режим «безопасного копирования»

Предложение `BEGIN BACKUP` предназначено для перевода базы данных в режим «безопасного копирования» («copy-safe»). Этот оператор «замораживает» основной файл базы данных, что позволяет безопасно делать резервную копию средствами файловой системы, даже если пользователи подключены и выполняют операции с данными. При этом все изменения, вносимые пользователями в базу данных, будут записаны в отдельный файл, так называемый дельта файл (delta file).

Предложение `END BACKUP` предназначено для перевода базы данных из режима «безопасного копирования» «copy-safe» в режим нормального функционирования. Этот оператор объединяет файл дельты с основным файлом базы данных и восстанавливает нормальное состояние работы, таким образом, закрывая возможность создания безопасных резервных копий средствами файловой системы.

6.5.4 Изменение набора символов по умолчанию

Предложение `SET DEFAULT CHARACTER SET` изменяет набор символов по умолчанию для базы данных. Это изменение не затрагивает существующие данные. Новый набор символов по умолчанию будет использоваться только в последующих DDL командах, кроме того для них будет использоваться сортировка по умолчанию для нового набора символов.

6.5.5 LINGER

Предложение `SET LINGER` позволяет установить задержку закрытия базы данных. Этот механизм позволяет ядру SuperServer, сохранять базу данных в открытом состоянии в течение некоторого времени после того как последние соединение закрыто, т.е. иметь механизм задержки закрытия базы данных. Это может помочь улучшить производительность и уменьшить издержки в случаях, когда база данных часто открывается и закрывается, сохраняя при этом ресурсы «разогретыми» до следующего открытия.

Предложение `DROP LINGER` удаляет задержку и возвращает базу данных к нормальному состоянию (без задержки). Эта команда эквивалентна установки задержки в 0.

Удаление LINGER не самое лучшее решение для временной необходимости его отключения для некоторых разовых действий, требующих принудительного завершения работы сервера. Утилита gfix имеет переключатель -NoLinger, который сразу закроет указанную базу данных, после того как последнего соединения не стало, независимо от установок LINGER в базе данных. Установка LINGER будет сохранена и нормально отработает в следующий раз.

6.5.6 Изменение привилегий выполнения по умолчанию

Предложение SET DEFAULT SQL SECURITY меняет поведение по умолчанию, которое определяет в контексте какого пользователя будет выполняться объект базы данных (процедура, функция, пакет, триггер, таблица). Ключевое слово INVOKER указывает, что объект выполняется с правами вызвавшего его пользователя. Задание ключевого слова DEFINER означает, что объект выполняется с правами к объектам базы данных его владельца (создателя). Изначально для базы данных стоит значение INVOKER.

6.5.7 Шифрование базы данных

Предложение ENCRYPT WITH шифрует базу данных с помощью указанного плагина шифрования. Шифрование начинается сразу после этого оператора и будет выполняться в фоновом режиме. Нормальная работа с базами данных не нарушается во время шифрования.

Процесс шифрования может быть проконтролирован с помощью поля MON$CRYPT_PAGE в таблице MON$DATABASE или просмотрен на странице заголовка базы данных с помощью gstat -e. gstat -h также будет предоставлять ограниченную информацию о состоянии шифрования.

Необязательное предложение KEY позволяет передать имя ключа для плагина шифрования. Что делать с этим именем ключа решает плагин.

Предложение DECRYPT дешифрует базу данных.

6.6 Удаление базы данных

Для удаления существующей базы данных, с которой выполнено в настоящий момент соединение, используется оператор SQL DROP DATABASE (листинг 6.6):

Листинг 6.6. Синтаксис оператора удаления существующей базы данных DROP DATABASE

```
DROP DATABASE;
```

Прежде чем удалять базу данных, с ней нужно соединиться. Оператор удаляет первичный, все вторичные файлы базы данных и все файлы оперативных копий (см. главу 7), связанные с этой базой данных.

Удалять базу данных может ее владелец, администратор и пользователь с привилегией DROP DATABASE.

Следует быть осторожным при использовании этого оператора. При удалении базы данных теряются все содержащиеся в ней данные и метаданные.

Удалить базу данных можно и обычными средствами операционной системы, просто удалив все файлы базы данных. Однако использование оператора DROP DATABASE гарантированно вызовет удаление всех связанных с базой данных файлов — первичного, всех вторичных файлов, а также файлов всех оперативных копий базы данных.

Пример. Следующие операторы позволяют удалить существующую базу данных. Вначале выполняется соединение с базой данных, затем эта база данных удаляется:

```
CONNECT 'localhost:D:\RedSoftDatabase\work.fdb'
USER 'wizard' PASSWORD 'master';
DROP DATABASE;
```

6.7 Создание примечаний для базы данных и объектов базы данных

Объекты базы данных и сама база данных могут содержать примечания. Это довольно удобное средство документирования процесса разработки создаваемой базы данных и ее объектов. Для этих целей используется оператор COMMENT. Синтаксис оператора представлен в листинге 6.7.

Листинг 6.7. Синтаксис оператора создания примечания для объектов базы данных COMMENT

```
COMMENT ON <объект> IS {'<текст>' | NULL}

<объект> ::= DATABASE
           | <базовый тип> <имя>
           | COLUMN <таблица>.<столбец>
           | [PROCEDURE | FUNCTION] PARAMETER [<пакет>.]<процедура>.<параметр>
           | {PROCEDURE | [EXTERNAL] FUNCTION} [<пакет>.]<процедура>

<базовый тип> ::= CHARACTER SET
                | COLLATION
                | DOMAIN
                | EXCEPTION
                | FILTER
                | GENERATOR
                | INDEX
                | PACKAGE
                | USER
                | ROLE
                | SEQUENCE
                | TABLE
                | TRIGGER
                | VIEW
```

Примечание можно создавать для любого объекта базы данных.

Добавить комментарий может администратор, владелец объекта, для которого добавляется комментарий, пользователь с привилегией ALTER ANY <тип объекта>.

Если задается ключевое слово DATABASE, то текст примечания создается именно для самой базы данных, иначе нужно указать базовый тип (объект метаданных), для которого будет создаваться примечание.

Если текст любого примечания задать в виде двух подряд идущих апострофов '', то это равносильно заданию NULL, то есть удалению существующего примечания объекта.

При создании базы данных можно также одновременно задать следующим оператором и текст

примечания. Например:

```
CREATE DATABASE 'localhost:D:\RedSoftDatabase\work.fdb'
USER 'wizard' PASSWORD 'master'
PAGE_SIZE = 16384
DEFAULT CHARACTER SET WIN1251;
COMMENT ON DATABASE IS 'Проверочная база данных';
```

Текст примечания любого объекта базы данных можно изменить в любое время, соединившись с базой данных и выполнив оператор `COMMENT`:

```
CONNECT 'localhost:D:\RedSoftDatabase\work.fdb'
USER 'wizard' PASSWORD 'master';
COMMENT ON DATABASE IS '<Новый текст примечания>';
```

Глава 7

Оперативные копии (SHADOW)

В РЕД База Данных существуют средства ведения оперативного копирования базы данных (`shadowing`). Оперативная копия (`shadow` — дословно тень), будучи созданной, содержит точную копию оригинальной базы данных. Когда создается оперативная копия, все изменения в базе данных тут же отражаются в оперативной копии. Если по различным причинам база данных станет недоступной (например, при сбое диска или при случайном удалении файлов базы данных), то автоматически происходит переключение работы клиентских программ на оперативную копию, которая станет рассматриваться клиентскими программами как исходная база данных.

Это средство относится только к текущим операциям с базой данных клиентских процессов, но не к новым подключениям клиентов. В случае поломки исходной базы данных дальнейшие действия администратора базы данных должны включать восстановление работоспособности начального дискового носителя базы данных и восстановление самой базы данных, возможно, с использованием данных оперативной копии. Только после этого возможно подключение новых клиентов к базе данных.

Исходя из назначения оперативных копий, их следует размещать на устройствах, отличных от устройств, где находятся файлы самой используемой базы данных.

Оперативная копия может быть создана в автоматическом режиме (ключевое слово `AUTO` в операторе создания оперативной копии — см. ниже) или в так называемом ручном режиме (ключевое слово `MANUAL`).

В случае автоматического режима (`AUTO`) в ситуации, когда сама оперативная копия по каким-либо причинам становится недоступной (например, сбой диска, случайное удаление файлов оперативной копии), работа клиентских программ продолжается обычным образом. При этом процесс ведения оперативного копирования для этой оперативной копии не осуществляется, он просто приостанавливается. Однако если существуют другие оперативные копии, заданные в базе данных, то процесс копирования для них продолжается.

Если же задан режим ручного копирования (`MANUAL`), то в случае недоступности файлов такой оперативной копии все клиентские процессы работы с базой данных прекращаются, пока администратор базы данных не отменит процесс копирования данных в эту «поломанную» оперативную копию. Для этого, как минимум, он должен удалить оперативную копию, используя оператор `DROP SHADOW`, и, при необходимости, создать новую оперативную копию с использованием оператора `CREATE SHADOW`.

Решение об использовании оперативных копий принимает администратор базы данных. Во многих случаях, когда важен процесс сохранения введенных клиентами данных, имеет смысл использовать оперативные копии, рассчитывая на то, что поломки диска, где хранится база данных, могут происходить достаточно часто. Оперативное копирование не занимает много времени и не требует затрат больших ресурсов вычислительной системы.

7.1 Создание оперативной копии

Для создания новой оперативной копии для базы данных, с которой выполнено в настоящий момент соединение, используется оператор `CREATE SHADOW`. Синтаксис представлен в листинге 7.1:

Листинг 7.1. Синтаксис оператора создания оперативной копии `CREATE SHADOW`

```
CREATE SHADOW <номер оперативной копии> [AUTO | MANUAL][CONDITIONAL]
              '<спецификация файла>' [LENGTH [=] <целое> [PAGE[S]]]
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
                [<вторичный файл>]...;

<вторичный файл> ::= FILE '<спецификация файла>'
                     [LENGTH [=] <целое> [PAGE[S]]]
                     [STARTING [AT [PAGE]] <целое>]
```

Создать оперативную копию может владелец базы данных, администратор и пользователь с привилегией `ALTER DATABASE`.

Оперативная копия начинает дублировать основную базу данных сразу в момент создания этой копии.

Номер оперативной копии — положительное число (не ноль), идентифицирующее набор файлов данной оперативной копии.

Если по каким либо причинам файл базы данных становится недоступным, то система преобразует тень в копию базы данных и переключается на неё. Теневая копия становится недоступной. Что будет дальше зависит от выбранного режима.

7.1.1 Режимы AUTO и MANUAL

При выборе альтернативы `AUTO` (значение по умолчанию) происходит следующее. Если файлы оперативной копии по различным причинам становятся недоступными, то завершаются все соединения с оперативной копией, удаляются все ссылки на эту оперативную копию. Работа с базой данных продолжается обычным образом без выполнения оперативного копирования в данную оперативную копию. Если существуют другие оперативные копии, то их использование продолжается обычным образом.

В случае `MANUAL`, если оперативная копия становится недоступной, то все попытки соединения с базой данных и обращения к ней будут вызывать сообщения об ошибках, пока оперативная копия не станет доступной или пока оперативная копия не будет удалена из базы данных администратором при помощи оператора `DROP SHADOW`. Администратор базы данных должен удалить ссылку на оперативную копию из базы данных, при необходимости удалить все файлы этой оперативной копии с диска и создать новую оперативную копию (если необходимо).

В случае, когда оперативная копия заменяет базу данных, можно указать новую оперативную копию, которая начнет выполнять функции оперативного копирования. Для этого нужно создать оперативную копию с ключевым словом `CONDITIONAL`. Это условная оперативная копия, которая заменяет бывшую активной перед этим оперативную копию, которая стала выполнять функции основной базы данных.

7.1.2 Файл оперативной копии

Спецификация файла оперативной копии — имя файла и полный к нему путь в соответствии с требованиями используемой операционной системы. Для Windows спецификация файла содержит имя устройства, путь к файлу и имя файла с его расширением. Имена файлов оперативной копии часто выбираются такими же, как и у файла базы данных, а для расширений имен файлов оперативных копий обычно применяются `shd, sh1, sh2` и т.д.

В момент создания оперативной копии на диске не должно быть файла с тем же именем, что и создаваемая оперативная копия.

Как и в случае с файлами базы данных оперативная копия может состоять из одного или нескольких файлов. Количество и размеры файлов оперативной копии никак не связаны с количеством и размерами первичного и/или вторичных файлов основной базы данных. База данных может состоять из нескольких файлов, а оперативная копия — только из одного и наоборот.

Для файлов теневой копии размер страницы устанавливается равным размеру страницы базы

данных и не может быть изменён.

7.1.3 LENGTH

Предложение LENGTH задает максимальный размер первичного или вторичного файла оперативной копии в страницах. Для единственного или последнего файла оперативной копии этот размер никак не влияет на величину используемой файлом памяти. Размер файла будет автоматически увеличиваться при необходимости до максимальной величины, которую обеспечивает используемая операционная система, или пока не будет исчерпано дисковое пространство носителя.

7.1.4 Вторичные файлы

Предложение STARTING AT задает номер страницы, с которой должен начинаться следующий файл копии. Когда предыдущий файл будет полностью заполнен данными в соответствии с заданным размером, система начнет помещать новые данные в следующий вторичный файл.

При создании многофайловой оперативной копии необходимо либо для предыдущего файла в списке указать предложение LENGTH, либо для последующего указывать предложение STARTING AT.

Нет возможности добавить новые вторичные файлы к существующей оперативной копии. В случае необходимости добавления дополнительных вторичных файлов или изменения их количественных характеристик нужно удалить существующую оперативную копию и заново ее создать, указав нужное количество вторичных файлов и их размеры.

Пример 1. Чтобы создать для базы данных однофайловую оперативную копию, нужно выполнить следующий оператор:

```
CONNECT 'localhost:D:\RedSoftDatabase\work.fdb'
   USER 'wizard' PASSWORD 'master';
create shadow 1 'd:\RedSoftDatabase\work.shd';
```

Пример 2. Для создания второй многофайловой оперативной копии следует выполнить операторы:

```
CONNECT 'localhost:D:\RedSoftDatabase\work.fdb'
   USER 'wizard' PASSWORD 'master';
create shadow 2 'd:\RedSoftDatabase\work.sh1'
   LENGTH = 5000 PAGES
   FILE 'd:\RedSoftDatabase\work.sh2';
```

Здесь будет создан второй набор оперативных копий, работающих одновременно с первым набором оперативных копий. Этот набор копий содержит два файла. Размер первого файла оперативной копии составляет 5000 страниц. Второй файл будет при необходимости увеличиваться до размеров, допустимых в используемой операционной системе или пока не будет исчерпано пространство на внешнем носителе.

7.2 Удаление оперативной копии

Для удаления существующей оперативной копии используется оператор SQL DROP SHADOW. Его синтаксис представлен в листинге 7.2:

Листинг 7.2. Синтаксис оператора удаления существующей оперативной копии DROP SHADOW

```
DROP SHADOW <номер оперативной копии> [{PRESERVE | DELETE} FILE];
```

Номер оперативной копии — положительное число, идентифицирующее набор файлов ранее созданной оперативной копии.

При удалении оперативной копии прекращается процесс дублирования данных в этой оперативной копии. Если указана опция `DELETE FILE` (по умолчанию), то будут также удалены и все связанные файлы с этой теневой копией. Если указана опция `PRESERVE FILE`, то файлы останутся не тронутыми. Это может быть полезно, если делать резервную копию с теневого файла.

Оператор завершается без ошибок и без каких-либо сообщений, даже если указанной оперативной копии у базы данных не существует.

Оперативная копия может быть удалена владельцем базы данных, администратором и пользователем с привилегией `ALTER DATABASE`.

Глава 8

Домены (DOMAIN)

Домен — объект реляционной базы данных, позволяющий описывать характеристики столбцов таблиц. При создании или изменении любой таблицы при описании столбцов можно ссылаться на существующий домен для копирования всех его характеристик в столбец таблицы. При копировании в столбец отдельные характеристики, описанные в домене, могут быть изменены и дополнены. Домены также можно использовать при описании локальных переменных и параметров в хранимых процедурах и в триггерах.

У домена есть обязательные и необязательные атрибуты. Тип данных является обязательным атрибутом. Необязательные атрибуты: значение по умолчанию, ограничение NOT NULL, условие CHECK, набор символов (для символьных типов данных и текстовых BLOB), порядок сортировки (для символьных типов данных).

При определении столбца на основе домена можно переопределить некоторые атрибуты, унаследованные от домена.

Таблица 8.1 — Правила переопределения атрибутов домена при определении столбца

Атрибут	Возможность переопределения	Описание
Тип данных	Нет	
Значение по умолчанию	Есть	
Набор символов	Есть	Можно использовать для восстановления значений базы данных по умолчанию для столбца.
Порядок сортировки	Есть	
Условие CHECK	Есть	Для добавления нового условия можно использовать предложения CHECK в операторах CREATE и ALTER на уровне таблицы.
Ограничение NOT NULL	Нет	Часто лучше не указывать NOT NULL при определении домена, а добавить его при определении столбца, если это необходимо.

8.1 Создание домена

Для создания нового домена в базе данных используется оператор CREATE DOMAIN. Синтаксис оператора представлен в листинге 8.1:

Листинг 8.1. Синтаксис оператора создания домена CREATE DOMAIN

```
CREATE DOMAIN <имя домена> [AS] <тип данных>
   [DEFAULT {<литерал> | NULL | <контекстная переменная>}]
   [NOT NULL]
   [CHECK (<условие домена>)]
   [CHARACTER SET <набор символов> [COLLATE <порядок сортировки>] ];
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
<тип данных> ::=
   <скалярный тип данных> | <тип BLOB> | <массив>

<скалярный тип данных> ::=
  SMALLINT | INT[EGER] | BIGINT | INT128
| REAL | FLOAT [(точность)] | DOUBLE PRECISION
| DECFLOAT [(точность)]
| BOOLEAN
| DATE
| TIME [{WITHOUT | WITH} TIME ZONE]
| TIMESTAMP [{WITHOUT | WITH} TIME ZONE]
| {DECIMAL | DEC | NUMERIC} [(точность [, масштаб])]
| {VARCHAR | {CHAR | CHARACTER} VARYING} (длина)
   [CHARACTER SET <набор символов>]
| {CHAR | CHARACTER} [(длина)] [CHARACTER SET <набор символов>]
| {NCHAR | NATIONAL {CHARACTER | CHAR}} VARYING (длина)
| {NCHAR | NATIONAL {CHARACTER | CHAR}} [(длина)]
| BINARY [(длина)]
| {VARBINARY | BINARY VARYING} (длина)

<тип BLOB> ::=
  BLOB [SUB_TYPE {<номер подтипа> | <имя подтипа>}]
  [SEGMENT SIZE <размер сегмента>] [CHARACTER SET <набор символов>]
| BLOB [(<размер сегмента> [, <номер подтипа>])]
| BLOB [(, <номер подтипа>)]

<массив> ::=
   <скалярный тип данных> <размерность массива>

<размерность массива> ::= '[' [m:]n [,[m:]n ...] ']'

<условие домена> ::= {
   <значение> <оператор сравнения> <значение>
 | <значение> [NOT] IN ({<значение> [, <значение> ...] | <поиск одного>})
 | <значение> [NOT] BETWEEN <значение> AND <значение>
 | <значение> [NOT] LIKE <шаблон> [ESCAPE <символ>]
 | <значение> [NOT] SIMILAR TO <значение> [ESCAPE <значение>]
 | <значение> IS [NOT] NULL
 | <значение> IS [NOT] DISTINCT FROM <значение>
 | <значение> <оператор сравнения> {ALL | SOME | ANY} (<поиск одного>)
 | [NOT] EXISTS (<поиск многих>)
 | [NOT] SINGULAR (<поиск многих>)
 | <значение> [NOT] CONTAINING <значение>
 | <значение> [NOT] STARTING [WITH] <значение>
 | (<условие домена>)
 | NOT <условие домена>
 | <условие домена> OR <условие домена>
 | <условие домена> AND <условие домена>
}

<значение> ::=
  VALUE [[<элемент массива>]]
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
  | <литерал>
  | <контекстная переменная>
  | <выражение>
  | NULL
  | NEXT VALUE FOR <имя генератора>
  | GEN_ID(<имя генератора>, <значение>)
  | CAST(<значение> AS <тип данных>)
  | (<выбор одного>)
  | <обычная встроенная функция> (<параметры>)
  | <агрегатная функция в операторе SELECT>
  | <функция UDF> [(<параметр> [, <параметр> ...])]

<оператор сравнения> ::=
    <> | != | ^= | ~= | = | < | > | <= | >=
  | !< | ^< | ~< | !> | ^> | ~>
```

Имя домена должно быть уникальным среди имен доменов базы данных. Имя не может превышать 63 символов.

В качестве типа домена может быть указан любой тип данных SQL.

Создавать домен может администратор и пользователь с привилегией CREATE DOMAIN.

8.1.1 Задание типа данных

Тип данных задается следующей синтаксической конструкцией (листинг 8.2):

Листинг 8.2. Синтаксис задания типа данных

```
<тип данных> ::=
   <скалярный тип данных> | <тип BLOB> | <массив>

<скалярный тип данных> ::=
    SMALLINT | INT[EGER] | BIGINT | INT128
  | REAL | FLOAT [(точность)] | DOUBLE PRECISION
  | DECFLOAT [(точность)]
  | BOOLEAN
  | DATE
  | TIME [{WITHOUT | WITH} TIME ZONE]
  | TIMESTAMP [{WITHOUT | WITH} TIME ZONE]
  | {DECIMAL | DEC | NUMERIC} [(точность [, масштаб])]
  | {VARCHAR | {CHAR | CHARACTER} VARYING} (длина)
     [CHARACTER SET <набор символов>]
  | {CHAR | CHARACTER} [(длина)] [CHARACTER SET <набор символов>]
  | {NCHAR | NATIONAL {CHARACTER | CHAR}} VARYING (длина)
  | {NCHAR | NATIONAL {CHARACTER | CHAR}} [(длина)]
  | BINARY [(длина)]
  | {VARBINARY | BINARY VARYING} (длина)

<тип BLOB> ::=
    BLOB [SUB_TYPE {<номер подтипа> | <имя подтипа>}]
    [SEGMENT SIZE <размер сегмента>] [CHARACTER SET <набор символов>]
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
  | BLOB [(<размер сегмента> [, <номер подтипа>])]
  | BLOB [(, <номер подтипа>)]

  <массив> ::=
     <скалярный тип данных> <размерность массива>

  <размерность массива> ::= '[' [m:]n [,[m:]n ...] ']'
```

Массивы

Если домен должен быть массивом, базовым типом может быть любой тип данных SQL, кроме BLOB и массива. Размерность массива указывается в квадратных скобках. Для каждой размерности массива одно или два целых числа определяют нижнюю и верхнюю границы диапазона его индексов. Нижняя граница является неявной (предполагается значение 1), и требуется указать только верхнюю. Для явного определения диапазона можно использовать два числа, разделенных двоеточием (':'), где второе число больше первого. Границы могут быть меньше нуля, при условии, что верхняя граница больше нижней.

Если массив многомерный, определения границ для каждой размерности должны быть разделены запятыми.

Размерности проверяются только в том случае, если массив действительно существует. Это означает, что сообщения об ошибках, связанных с размерностью, не будут возвращены, если выбор определенного элемента ничего не дает или если поле массива является NULL.

Массив, как и тип данных BLOB, не хранится непосредственно в строке таблицы. Строка содержит лишь ссылку на страницу базы данных, где располагаются элементы массива. Элементы массива располагаются в других страницах базы данных.

Строковые типы

С помощью предложения CHARACTER SET можно указать набор символов для типов CHAR, VARCHAR и BLOB SUB_TYPE TEXT. Если набор символов не указан, то будет использоваться набор символов, указанный для базы данных DEFAULT CHARACTER SET. Если для базы данных не определён набор символов по умолчанию, то при создании символьного домена по применится набор символов NONE.

При использовании NONE символьные данные хранятся и извлекаются в том виде, в котором они были представлены. В столбец, основанный на таком домене, можно добавить данные в любой кодировке, но невозможно добавить эти данные в столбец с другой кодировкой. Поскольку транслитерация между исходной и конечной кодировками не выполняется, возможны ошибки.

8.1.2 Значение по умолчанию

Необязательное предложение DEFAULT определяет значение по умолчанию для домена. Это значение будет добавлено в столбец, основанный на этом домене, при выполнении оператора INSERT, если в операторе DML для него не указано значение.

Значение по умолчанию не используется при выполнении оператора UPDATE. Если в этом операторе пользователь не укажет значение для конкретного столбца, то это значение просто не изменяется.

Локальные переменные и параметры в модулях PSQL, которые ссылаются на этот домен, будут инициализированы значением по умолчанию.

Значением по умолчанию может быть литерал, значение NULL. Литералом может быть любая самоопределенная константа соответствующего типа, предварительно определенный литерал или контекстная переменная. Если значение по умолчанию не указано, то подразумевается NULL.

8.1.3 Значение NOT NULL

Предложение NOT NULL указывает, что столбцу, основанному на этом домене, не может присваиваться пустое значение ни в операторе INSERT, ни в операторе UPDATE. Это предложение является обязательным, если домен будет использован для создания столбца, входящего в состав первичного ключа таблицы.

Любая попытка поместить в такой столбец пустое значение в операторе INSERT или изменить на NULL существующее значение в операторе UPDATE приводит к исключению базы данных.

Предложение NOT NULL всегда нужно явно задавать для столбцов первичного ключа, даже если на основании других условий домена (предложение CHECK, см. ниже) ему не может быть присвоено пустое значение. Для других доменов, используемых в иных столбцах таблиц, это значение может устанавливаться в соответствии с требованиями конкретной предметной области. Например, в таблице, содержащей данные о человеке, можно для домена, используемого при создании столбца, содержащего фамилию человека, указать NOT NULL.

8.1.4 Условие домена

Необязательное предложение CHECK определяет условия, которым должны удовлетворять значения столбцов таблицы или переменных, наследуемых от домена. Условие должно быть заключено в круглые скобки. Предложение CHECK нельзя использовать с доменами типа BLOB.

Условие - это логическое выражение (также называемое предикатом), которое может возвращать результаты TRUE, FALSE и UNKNOWN. Условие считается выполненным, если предикат возвращает значение TRUE или UNKNOWN. Если предикат возвращает FALSE, то условие не выполнено. Подробнее обо всех предикатах рассказано в главе 2.

Условие домена может быть достаточно сложным. Его синтаксис показан в листинге 8.3.

Листинг 8.3. Синтаксис задания условий домена

```
<условие домена> ::= {
   <значение> <оператор сравнения> <значение>
 | <значение> [NOT] IN ({<значение> [, <значение> ...] | <поиск одного>})
 | <значение> [NOT] BETWEEN <значение> AND <значение>
 | <значение> [NOT] LIKE <шаблон> [ESCAPE <символ>]
 | <значение> [NOT] SIMILAR TO <значение> [ESCAPE <значение>]
 | <значение> IS [NOT] NULL
 | <значение> IS [NOT] DISTINCT FROM <значение>
 | <значение> <оператор сравнения> {ALL | SOME | ANY} (<поиск одного>)
 | [NOT] EXISTS (<поиск многих>)
 | [NOT] SINGULAR (<поиск многих>)
 | <значение> [NOT] CONTAINING <значение>
 | <значение> [NOT] STARTING [WITH] <значение>
 | (<условие домена>)
 | NOT <условие домена>
 | <условие домена> OR <условие домена>
 | <условие домена> AND <условие домена>
}

<значение> ::=
  VALUE [[<элемент массива>]]
| <литерал>
| <контекстная переменная>
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
  | <выражение>
  | NULL
  | NEXT VALUE FOR <имя генератора>
  | GEN_ID(<имя генератора>, <значение>)
  | CAST(<значение> AS <тип данных>)
  | (<выбор одного>)
  | <обычная встроенная функция> (<параметры>)
  | <агрегатная функция в операторе SELECT>
  | <функция UDF> [(<параметр> [, <параметр> ...])]

<оператор сравнения> ::=
    <> | != | ^= | ~= | = | < | > | <= | >=
  | !< | ^< | ~< | !> | ^> | ~>
```

Выбор одного — это оператор `SELECT`, возвращающий в точности одно значение одного столбца, получаемое из таблицы, представления или в качестве выходного параметра хранимой процедуры выбора. Пустое значение недопустимо.

Поиск одного — оператор `SELECT`, возвращающий произвольное количество значений одного столбца. Здесь возможно пустое значение.

Поиск многих — оператор `SELECT`, возвращающий ноль или произвольное количество значений нескольких столбцов.

Ключевое слово VALUE

В значении нельзя использовать имена доменов или столбцов таблиц. В качестве одного из элементов значения может использоваться ключевое слово `VALUE`.

Ключевое слово `VALUE` является заменителем имени столбца, который при создании таблицы будет основан на данном домене. Обычно в условиях домена `VALUE` помещается в левой части оператора, но допустимо также помещение его в качестве, например, элемента выражения, и в правую часть. К этому ключевому слову можно применять функцию `UPPER`, переводящую все буквы в добавляемом/изменяемом значении столбца, основанного на данном домене в верхний регистр. Допустимо также использование функции преобразования типов данных `CAST`, функции выделения подстроки `SUBSTRING`, функции удаления начальных и конечных пробелов `TRIM` и других встроенных функций.

Если домен описывает массив, то, как обычно, после ключевого слова `VALUE` в квадратных скобках нужно указать индекс (индексы) конкретного элемента массива.

Любое выражение может содержать ключевое слово `VALUE`.

В варианте `VALUE <оператор> <значение>` помещаемое значение в столбец, основанный на этом домене, сравнивается с некоторым литералом или выражением. Чтобы значение было помещено в столбец, сравнение должно давать значение «истина».

Пример 1.

Пример описания условия домена для числового столбца:

```
CHECK (VALUE >= 18)
```

Обратите внимание, что использование этого условия неявно не допускает возможности помещения в соответствующий столбец пустого значения.

Пример 2.

Путь требуется проверка того, что первый символ во вводимом строковом значении должен равняться второму. Здесь можно использовать функцию выделения подстроки:

```
CHECK (SUBSTRING(VALUE FROM 1 FOR 1) = SUBSTRING(VALUE FROM 2 FOR 1));
```

Если же при этом вводятся буквы в различных регистрах и требуется проверка на равенство независимо от того, строчные это или прописные буквы, то можно дополнительно использовать и более сложную конструкцию, включив функцию UPPER:

```
CHECK (SUBSTRING(UPPER(VALUE) FROM 1 FOR 1) =
       SUBSTRING(UPPER(VALUE) FROM 2 FOR 1));
```

С тем же результатом в предыдущем выражении можно вместо встроенной функции UPPER использовать функцию LOWER.

Следует помнить, что в некоторых вариантах проверки условия CHECK, если для соответствующего столбца допустимо и пустое значение, то проверка на NULL должна быть выполнена как отдельная часть проверки условия. Иначе в некоторых случаях будет возвращено значение UNKNOWN, что вызовет исключение базы данных. Проверку на пустое значение нужно соединить операцией дизъюнкции (ключевое слово OR) с основной содержательной проверкой:

```
CHECK ((VALUE IS NULL) OR (<остальные виды проверок>))
```

Пример 3.

Создание домена, который может принимать значения 'Да' и 'Нет'.

```
CREATE DOMAIN D_BOOLEAN AS CHAR(3)
CHECK (VALUE IN ('Да' , 'Нет' ));
```

Пример 4.

Создание домена со значением по умолчанию.

```
CREATE DOMAIN D_DATE AS DATE
DEFAULT CURRENT_DATE
NOT NULL;
```

8.1.5 Ключевое слово COLLATE

Необязательное предложение COLLATE позволяет указать порядок сортировки для домена основаного на строковом типе данных, включая BLOB с текстовым подтипом.

Если сортировка не указана, то будет использоваться сортировка по умолчанию для набора символов, указанного в момент создания домена.

8.2 Изменение домена

Для изменения характеристик существующего в базе данных домена используется оператор ALTER DOMAIN. Синтаксис оператора показан в листинге 8.4.

Листинг 8.4. Синтаксис оператора изменения домена ALTER DOMAIN

```
ALTER DOMAIN <имя>
   [TO <новое имя>]
   [{ SET DEFAULT {<литерал> | NULL | <контекстная переменная>}
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
     | DROP DEFAULT}]
    [{ SET | DROP} NOT NULL]
    [{ ADD [CONSTRAINT] CHECK (<условие домена>)
     | DROP CONSTRAINT}]
    [TYPE <тип данных> [CHARACTER SET <набор символов> [COLLATE <порядок сортировки>
]]];
```

В одном операторе `ALTER DOMAIN` можно выполнить любое количество изменений домена.

Изменить существующий домен может владелец домена (его создатель), пользователь с административными привилегиями или пользователь с привилегией `ALTER ANY DOMAIN`.

Изменение имени

Имя домена можно изменять, даже если на этом домене основаны столбцы уже существующих в базе данных таблиц или внутренние переменные хранимых процедур и триггеров. При этом для каждого столбца и каждой переменной, основанной на домене, у которого меняется имя, просто изменяется ссылка на имя базового домена.

Значение по умолчанию

Предложение `SET DEFAULT` позволяет установить новое значение по умолчанию. Если у домена уже было задано значение по умолчанию, то создание нового значения по умолчанию не требует явного удаления предыдущего значения.

Предложение `DROP DEFAULT` удаляет существующее значение по умолчанию. Значением по умолчанию в этом случае неявно становится значение `NULL`.

Условие CHECK

Предложение `ADD [CONSTRAINT] CHECK` добавляет условие домена. Если у домена уже существует условие, то вначале его нужно удалить при помощи предложения `DROP CONSTRAINT` иначе вы получите сообщение об ошибке.

Предложение `DROP CONSTRAINT` удаляет существующее ограничение `CHECK` домена. Если домен не содержит ограничения с таким именем, то выполнение подобного оператора не вызовет сообщения об ошибке.

Изменение типа домена

Тип данных домена можно изменить при помощи предложения `TYPE`. Изменяет тип данных домена на другой, совместимый. Система запретит любое изменение типа, которое может привести к потере данных. Например, если количество символов в новом типе меньше, чем в существующем.

При изменении атрибутов домена существующий `PSQL` может стать невалидным. Информация о том, что модуль `PSQL` стал невалидным отразится в поле `RDB$VALID_BLR` в системных таблицах `RDB$PROCEDURES`, `RDB$FUNCTIONS` и `RDB$TRIGGERS`. `RDB$VALID_BLR` устанавливается в 0, если код стал невалидным в результате изменения домена.

```
create domain d1 varchar(30) character set UTF8 collate UNICODE;
alter domain d1 type varchar(30) character set UTF8 collate UCS_BASIC;
```

Ограничение NOT NULL

Предложение SET NOT NULL устанавливает ограничение NOT NULL для домена. В этом случае для переменных и столбцах базирующихся на домене значение NULL не допускается.

Добавление ограничения NOT NULL к существующему домену подвергнет все столбцы, использующие этот домен, полной проверке данных, поэтому убедитесь, что в столбцах нет значений NULL, прежде чем приступать к изменению.

Предложение DROP NOT NULL удаляет ограничение NOT NULL для домена. Явное ограничение NOT NULL на столбце, основанном на домене, перекрывает характеристику домена. В этом случае снятие ограничения NOT NULL с домена не распространится на столбец.

Пример изменения домена

Пример. В этом примере выполняются радикальные изменения в описании характеристик домена. Все это делается в одном операторе:

```
ALTER DOMAIN D099
  DROP DEFAULT
  SET DEFAULT USER
  DROP CONSTRAINT
  ADD CONSTRAINT
     CHECK (SUBSTRING(UPPER(VALUE) FROM 1 FOR 1) =
            SUBSTRING(UPPER(VALUE) FROM 2 FOR 1));
```

Здесь происходит удаление значения по умолчанию, а затем создается новое, USER — имя пользователя, соединенного в настоящий момент с базой данных. Для нового значения по умолчанию можно было бы указать и контекстную переменную CURRENT_USER. Результат будет точно таким же.

Чтобы заменить существующее условие домена вначале нужно выполнить удаление старого условия, после этого добавляется новое условие, в котором требуется (в нашем примере) равенство первого и второго символа в строковом данном.

8.3 Удаление домена

Для удаления домена используется оператор DROP DOMAIN. Его синтаксис представлен в листинге 8.5.

Листинг 8.5. Синтаксис оператора удаления домена DROP DOMAIN

```
DROP DOMAIN <имя домена>;
```

Оператор DROP DOMAIN удаляет существующий в базе данных домен. Невозможно удалить домен, если на него ссылаются столбцы таблиц базы данных или он используется в каком-либо модуле PSQL. Чтобы удалить используемый домен, необходимо удалить все столбцы, ссылающиеся на него, и удалить все ссылки на домен из модулей PSQL.

Удалить существующий домен может владелец домена (его создатель), пользователь с административными привилегиями или пользователь с привилегией DROP ANY DOMAIN.

Пример. Чтобы удалить домен CODCOUNTRY, нужно выполнить оператор:

```
DROP DOMAIN CODCOUNTRY;
```

Поскольку в базе данных, которая была здесь описана, существуют столбцы, ссылающиеся на этот домен, такой оператор не может быть выполнен без сообщений об ошибке. Предварительно нужно удалить во всех таблицах ссылки на этот домен.

8.4 Примечание домена

Для существующего домена вы можете создать комментарий, используя оператор `COMMENT ON DOMAIN` следующего вида (см. листинг 8.6):

Листинг 8.6. Синтаксис оператора примечания домена `COMMENT ON DOMAIN`

```
COMMENT ON
   DOMAIN <имя домена> IS {'<текст примечания>' | NULL};
```

Текст примечания любого домена можно изменять произвольное количество раз при выполнении оператора `COMMENT ON DOMAIN`. Значение `NULL` удаляет существующее примечание. Примечание домена может служить средством документирования разрабатываемой программной системы.

Выполнить оператор `COMMENT ON DOMAIN` могут администраторы, владельцы домена или пользователи с привилегией `ALTER ANY DOMAIN`.

Глава 9

Таблицы (TABLE)

Таблица — наиболее важный и сложный объект реляционной базы данных. В таблицах хранятся все обрабатываемые клиентскими программами данные базы данных. Все строки одной таблицы имеют одинаковую структуру. Количество строк в таблице произвольное. Таблица должна содержать хотя бы один столбец. Обрабатываемые (пользовательские) данные хранятся в таблицах, создаваемых пользователем при помощи оператора CREATE TABLE и изменяемых оператором ALTER TABLE. Системные данные (метаданные, описывающие объекты базы данных) хранятся в системных таблицах, которые создаются автоматически при первоначальном создании базы данных. База данных может содержать не более 32640 пользовательских таблиц.

9.1 Создание таблиц

Таблица создается оператором CREATE TABLE. Синтаксис оператора представлен в листинге 9.1.

Листинг 9.1. Синтаксис оператора создания таблицы CREATE TABLE

```
CREATE [GLOBAL TEMPORARY] TABLE <имя таблицы>
   [EXTERNAL [FILE] '<спецификация файла>' [ADAPTER 'CSV']]
   (<определение столбца> [, { <определение столбца> | <ограничение таблицы>}...])
   [ON COMMIT {DELETE | PRESERVE} ROWS]
   [SQL SECURITY {DEFINER | INVOKER}];
   [[IN] TABLESPACE {<имя табличного пространства> | PRIMARY}];

<определение столбца> ::= { <опр-е обычного столбца>
                          | <опр-е вычисляемого столбца>
                          | <опр-е идентификационного столбца> }

<определение обычного столбца> ::=
             <имя столбца> { <тип данных> | <имя домена>}
             [DEFAULT {<литерал> | NULL | <контекстная переменная>}]
             [NOT NULL]
             [<ограничение столбца>]
             [COLLATE <порядок сортировки>]

<определение вычисляемого столбца> ::=
             <имя столбца> [<тип данных>]
             {COMPUTED [BY] | GENERATED ALWAYS AS} (<выражение>)

<определение идентификационного столбца> ::=
       <имя столбца> [<тип данных>]
       GENERATED {ALWAYS|BY DEFAULT} AS IDENTITY [(<опции ав.> [<опции ав.>])]
       [<ограничение столбца>]

<опции автоинкремента> ::= START WITH <начальное значение>
                         | INCREMENT [BY] <приращение>

<ограничение столбца> ::=
  [CONSTRAINT <имя ограничения>]
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
  { UNIQUE [<предложение USING>] [[IN] TABLESPACE {<имя табл.пространства>|PRIMARY}]
  | PRIMARY KEY [<предложение USING>] [[IN] TABLESPACE {<имя табл.прос-ва>|PRIMARY}]
   | REFERENCES <имя таблицы> [(<имя столбца>)]
      [ON DELETE { NO ACTION | CASCADE | SET DEFAULT | SET NULL }]
      [ON UPDATE { NO ACTION | CASCADE | SET DEFAULT | SET NULL }]
      [<предложение USING>] [[IN] TABLESPACE { <имя табл. пространства> | PRIMARY}]
   | CHECK (<условие столбца>)
   }

<ограничение таблицы> ::=
   [CONSTRAINT <имя ограничения>]
   { UNIQUE (<столбец> [, <столбец>...]) [<предложение USING>]
        [[IN] TABLESPACE { <имя табличного пространства> | PRIMARY}]
   | PRIMARY KEY (<столбец> [, <столбец>...]) [<предложение USING>]
        [[IN] TABLESPACE { <имя табличного пространства> | PRIMARY}]
   | FOREIGN KEY (<столбец> [, <столбец>...])
      REFERENCES <имя таблицы> [(<столбец> [, <столбец>...])]
      [ON DELETE { NO ACTION | CASCADE | SET DEFAULT | SET NULL }]
      [ON UPDATE { NO ACTION | CASCADE | SET DEFAULT | SET NULL }]
      [<предложение USING>] [[IN] TABLESPACE { <имя табл. пространства> | PRIMARY}]
   | CHECK (<условие столбца>)
}

<предложение USING> ::= USING [ASC[ENDING] | DESC[ENDING]] INDEX <имя индекса>
```

Имя таблицы должно быть уникальным среди имен таблиц базы данных, хранимых процедур и представлений, описанных в этой базе данных.

Столбец должен иметь имя, уникальное только в данной таблице. В других таблицах могут присутствовать столбцы с тем же именем.

Таблица может содержать, по меньшей мере, один столбец и практически произвольное количество ограничений столбцов и ограничений таблицы. Описания столбцов и ограничений одной таблицы заключаются в круглые скобки и отделяются друг от друга запятыми.

Создать новую таблицу может администратор и пользователь с привилегией `CREATE TABLE`.

Пользователь, создавший таблицу, становится её владельцем.

9.1.1 Глобальные временные таблицы (GTTs)

Глобальные временные таблицы (Global Temporary Tables, GTTs) так же, как и обычные таблицы, являются постоянными метаданными, но данные в них ограничены по времени существования транзакцией (значение по умолчанию) или соединением с БД. Каждая транзакция или соединение имеет свой собственный экземпляр GTT с данными, изолированный от всех остальных. Экземпляры создаются только при условии обращения к GTT, и данные в ней удаляются при подтверждении транзакции или отключении от БД. Для изменения или удаления метаданных GTT можно использовать конструкции `ALTER TABLE` и `DROP TABLE`. Синтаксис создания временной таблицы представлен ниже:

Листинг 9.2. Синтаксис оператора создания глобальной временной таблицы

```
CREATE GLOBAL TEMPORARY TABLE <имя таблицы>
   (<определение столбца> [, {<определение столбца> | <ограничение таблицы>}...])
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
    [ON COMMIT {DELETE | PRESERVE} ROWS]
    [SQL SECURITY {DEFINER | INVOKER}];
```

Если в операторе создания глобальной временной таблицы указано необязательное предложение `ON COMMIT DELETE ROWS`, то будет создана GTT транзакционного уровня (по умолчанию). При указании предложения `ON COMMIT PRESERVE ROWS` будет создана GTT уровня соединения с базой данных.

Предложение `EXTERNAL [FILE]` нельзя использовать для глобальной временной таблицы.

Глобальные временные таблицы имеют ряд ограничений:

- GTT и обычные таблицы не могут ссылаться друг на друга;
- GTT уровня соединения (`PRESERVE ROWS`) не могут ссылаться на GTT транзакционного уровня (`DELETE ROWS`);
- Уничтожения экземпляра GTT в конце своего жизненного цикла не вызывает срабатывания триггеров до/после удаления
- Ограничения домена не могут ссылаться на временные таблицы.

В системном каталоге временные таблицы отличаются друг от друга и от постоянных таблиц значением типа `RDB$RELATION_TYPE` в системной таблице `RDB$RELATIONS`:

- GTT уровня соединения (`PRESERVE ROWS`) имеет тип `RDB$RELATION_TYPE = 4`
- GTT транзакционного уровня (`DELETE ROWS`) имеет тип `RDB$RELATION_TYPE = 5`

Пример создания глобальной временной таблицы уровня соединения:

```
CREATE GLOBAL TEMPORARY TABLE MyConnGTT (
   id INTEGER NOT NULL PRIMARY KEY,
   txt VARCHAR(32),
   ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP )
ON COMMIT PRESERVE ROWS;
```

Пример создания глобальной временной таблицы уровня транзакции, ссылающейся внешним ключом на глобальную временную таблицу уровня соединения:

```
CREATE GLOBAL TEMPORARY TABLE MyTrGTT (
   id INTEGER NOT NULL PRIMARY KEY,
   parent_id INT NOT NULL REFERENCES MyConnGTT(id),
   txt VARCHAR(32),
   ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP );
```

9.1.2 Использование внешних файлов

Сама таблица может и не храниться в базе данных, все ее строки могут помещаться в отдельный текстовый файл, находящийся вне базы данных. Для этого используется предложение `EXTERNAL [FILE]` в операторе создания таблицы. Спецификация внешнего файла должна содержать имя файла с его расширением и полный путь к этому файлу с учетом требований используемой операционной системы.

СУБД РЕД База Данных поддерживает два формата внешних файлов: формат «строк» с фиксированной длиной и `.csv` формат.

Столбцы таблицы, которые хранятся во внешних файлах, могут быть любого типа данных, кроме `BLOB`. Недопустимо также использование массивов с любым типом данных.

Над таблицей, хранящейся во внешнем файле, допустимы только операции выборки (`SELECT`) данных и добавления новых строк (`INSERT`). Операторы `INSERT` добавляют в конец существующего внешнего файла новые строки. Для `.csv` формата допустима только выборка данных. Операции же

изменения существующих данных (`UPDATE`) или удаления строк такой таблицы (`DELETE`) не могут быть выполнены. Попытки их выполнения вызовут исключения базы данных.

Внешняя таблица не может содержать ограничений первичного, внешнего и уникального ключа. Для полей такой таблицы невозможно создать индексы. Для внешней таблицы нельзя указать табличное пространство.

Возможность использования для таблиц внешних файлов зависит от установки значения параметра `ExternalFileAccess` в файле конфигурации `firebird.conf`. По умолчанию этот параметр имеет значение `None`, что запрещает использование для таблиц любой базы данных внешних файлов. Если параметр `ExternalFileAccess` содержит `Restrict`, то файл внешней таблицы должен находится в одном из каталогов, указанных в качестве аргумента `Restrict`.

Файлы с фиксированной длиной строк

Внешняя таблица, находящаяся в таком файле, имеет формат «строк» с фиксированной длиной. Нет никаких разделителей полей: границы полей и строк определяются максимальными размерами в байтах в определении каждого поля. Это необходимо помнить и при определении структуры внешней таблицы, и при проектировании входного файла для внешней таблицы, в которую должны импортироваться данные из другого приложения.

Если при обращении к внешней таблице РЕД База Данных не находит файла, то она создаёт его при первом обращении. Последующие операторы `INSERT` добавляют в конец существующего файла новые записи. Если записанные перед созданием таблицы данные в этом файле не соответствуют по структуре создаваемой таблице, то в дальнейшем при использовании таблицы обязательно возникнут проблемы.

Самым полезным типом данных для столбцов внешних таблиц является тип `CHAR` с фиксированной длиной, длина должна подходить под данные с которыми необходимо работать. Числовые типы и даты легко преобразуются в них.

Если данные читаются базой данных РЕД Базы Данных, то в то же время они могут оказаться нераспознаваемыми дня внешних приложений и являться для них «абракадаброй».

Конечно, существуют способы манипулирования типами данных так, чтобы создавать выходные файлы из РЕД Базы Данных, которые могут быть непосредственно прочитаны как входные файлы в других приложениях, используя хранимые процедуры с использованием внешних таблиц или без них. Описания этих методов выходит за рамки данного руководства. Здесь мы приведём лишь некоторые рекомендации и советы для создания и работы с простыми текстовыми файлами, поскольку внешняя таблица часто используется как простой способ для создания или чтения транзакционно-независимого журнала. Эти файлы могут быть прочитаны в оффлайн режиме текстовым редактором или приложением аудита.

Как правило, внешние файлы более удобны если строки разделены разделителем, в виде последовательности "новой строки", которая может быть распознана приложением на предназначенной платформе. Для Windows — это двухбайтная CRLF последовательность: возврат каретки (ASCII код 13) и перевод строки (ASCII код 10). Для POSIX — LF обычно самодостаточен, в некоторых MacOS X приложениях она может быть LFCR.

Пример.

Создадим внешнюю таблицу, которая содержит всего два столбца: метку времени и текстовое сообщение. Третий столбец хранит разделитель строки.

Существуют различные способы для автоматического заполнения столбца разделителя. В нашем примере это сделано с помощью `BEFORE INSERT` триггера и встроенной функции `ASCII_CHAR`.

```
CREATE TABLE ext_log
EXTERNAL FILE 'd:\externals\log_me.txt' (
   stamp CHAR(38),
   message CHAR(100),
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
    crlf CHAR(2) -- Для Windows
);
```

Теперь создадим триггер, для автоматического сохранения метки времени и разделителя строки, каждый раз когда сообщение записывается в таблицу:

```
CREATE TRIGGER bi_ext_log FOR ext_log
ACTIVE BEFORE INSERT
AS BEGIN
   IF (NEW.stamp IS NULL) THEN
       NEW.stamp = CAST (CURRENT_TIMESTAMP AS CHAR(38));
   NEW.crlf = ASCII_CHAR(13) || ASCII_CHAR(10);
END!
```

Вставка некоторых записей (это может быть сделано в обработчике исключения):

```
INSERT INTO ext_log (message) VALUES('Shall I compare thee to a summer''s day?');
INSERT INTO ext_log (message) VALUES('Thou art more lovely and more temperate');
```

Содержимое внешнего файла:

```
2024-06-03 10:11:05.0200 Europe/Moscow Shall I compare thee to a summer's day?
2024-06-03 10:11:05.0200 Europe/Moscow Thou art more lovely and more temperate
```

Файлы формата CSV

В СУБД РЕД База Данных реализован CSV адаптер внешних таблиц, позволяющий импортировать данные из файлов CSV. Использовать CSV-файлы для создания внешних таблиц можно с помощью предложения `EXTERNAL [FILE] '<спецификация файла>' ADAPTER 'CSV'` в операторе создания таблицы.

Как уже было сказано, для внешней таблицы в `.csv` формате допустима только операция выборки данных. Для редактирования CSV-файлов нужно использовать сторонние программы.

Каждая строка в CSV файле соответствует строке таблицы. Пустые строки игнорируются. В качестве разделителя значений полей используется запятая (',') без пробелов.

Количество значений в строках файла, разделенных запятыми, может не соответствовать количеству полей таблицы. Если значений меньше, то оставшиеся поля будут установлены в `NULL`. Если значений больше, то лишние значения проигнорируются. Если значение пропущено (например, `value1, value2,,value4`), то соответствующее поле также будет установлено в `NULL`.

Если значение по какой-либо причине не может быть сконвертировано в требуемый тип поля, то будет выброшено исключение.

Значения, содержащие зарезервированные символы (двойная кавычка, запятая, новая строка) обрамляются двойными кавычками (например, `value1,"one,two,three",value3`). Если в значении встречаются кавычки — они представляются в файле в виде двух кавычек подряд (например, `value1, "one,""two"",three",value3`).

Пример.

Пусть в CSV файле содержатся следующие строки:

```
01/03/1997,TESCO,"EVERY
LITTLE HELPS"
10/05/1967,M&M,"MELTS IN YOUR MOUTH, NOT IN YOUR HANDS"
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
06/23/1954,Disneyland,"I'm going to ""Walt Disney World""!"
07/15/1934,,JUST DO IT,aaa,bbb
,
```

Пример использования CSV адаптера:

```
create table CSV_EXT external 'D:\externals\table.csv' adapter 'CSV' (
   Sdate DATE,
   Company VARCHAR(14),
   Slogan VARCHAR(50)
);
```

Выборка из таблицы будет иметь такой вид:

```
SELECT * FROM CSV_EXT;

SDATE      COMPANY        SLOGAN
========== ============== ======================================
1997-01-03  TESCO          EVERY
                           LITTLE HELPS
1967-10-05  M&M            MELTS IN YOUR MOUTH, NOT IN YOUR HANDS
1954-06-23  Disneyland     I'm going to "Walt Disney World"!
1934-07-15  <null>         JUST DO IT
<null> <null>         <null>
```

9.1.3 Задание типа данных

Подробное описание типов данных, допустимых операций преобразования и других встроенных в SQL функций работы с данными см. в главе 3 и в Приложение З. Синтаксис задания типа данных столбца таблицы показан в листинге 9.3.

Листинг 9.3. Синтаксис задания типа данных столбца таблицы

```
<тип данных> ::= {
     {SMALLINT | INTEGER | BIGINT} [<размерность массива>]
   | BOOLEAN [<размерность массива>]
   | {FLOAT | DOUBLE PRECISION} [<размерность массива>]
   | DECFLOAT[({16 | 34})]
   | {DATE | TIME | TIMESTAMP} [<размерность массива>]
   | {DECIMAL | NUMERIC} [(<точность> [ ,<масштаб>])] [<размерность массива>]
   | {CHAR | CHARACTER | CHARACTER VARYING | VARCHAR} [(<размер>)]
       [<размерность массива>] [CHARACTER SET <набор символов>]
   | {NCHAR | NATIONAL CHARACTER | NATIONAL CHAR} [VARYING]
       [(<размер>)] [<размерность массива>]
   | BLOB [SUB_TYPE { <номер подтипа> | <имя подтипа>}]
       [SEGMENT SIZE <длина сегмента>] [CHARACTER SET <набор символов>]
   | BLOB [(<размер сегмента> [, <номер подтипа>])]
}

<размерность массива> ::= [[<целое 1>:]<целое 2> [, [<целое 1>:]<целое 2>...]]
```

Для столбца с любым типом данных, кроме BLOB, можно указать размерность массива, если этот

столбец является массивом. Для массива задается начальный номер элемента в массиве (положительное число «целое 1») и через двоеточие последний номер элемента («целое 2»). Если указано только одно число, то оно означает последний номер в элементе массива, а первым номером считается 1. Если массив многомерный, то через запятую указываются и другие пары элементов. Размерность задается в квадратных скобках.

При описании символьного столбца с типами `CHAR, VARCHAR` и `BLOB` с подтипом `TEXT` можно в предложении `CHARACTER SET` указать набор символов, если требуется набор, отличный от набора символов по умолчанию, установленного для всей базы данных. Если же при создании базы данных не был указан набор символов, то по умолчанию принимается набор символов `NONE`. В этом случае данные хранятся и извлекаются, так как они были поданы. В столбец можно загружать данные в любой кодировке, но невозможно загрузить эти данные в столбец с другой кодировкой. Транслитерация между исходными и конечными кодировками не выполняется, что может приводить к ошибкам.

Кроме того, в предложении `COLLATE` можно задать и порядок сортировки (для типа данных `BLOB` использование `COLLATE` недопустимо). Если порядок сортировки не указан, то по умолчанию принимается порядок сортировки по умолчанию для указанного набора символов.

Для типа данных `BLOB` можно указывать подтип (`SUB_TYPE`) и размер сегмента (`SEGMENT SIZE`). Существует два варианта синтаксиса для задания подтипа и размера сегмента:

```
BLOB [SUB_TYPE { <номер подтипа> | <имя подтипа>}]
     [SEGMENT SIZE <длина сегмента>] [CHARACTER SET <набор символов>]
```

и

```
BLOB [(<размер сегмента> [, <номер подтипа>])]
```

Размер сегмента задается в байтах. Он не может превышать 65535.

9.1.4 Использование ссылки на домен

Если вместо типа данных задается имя домена, то все его характеристики копируются для этого столбца. Если далее в описании столбца заданы и другие характеристики, то они заменяют скопированные характеристики из домена.

Если определение столбца основано на домене, оно может включать новое значение по умолчанию, дополнительные ограничения `CHECK`, предложение `COLLATE`, которые перекрывают значения указанные при определении домена. Определение такого столбца может включать дополнительные ограничения столбца, например `NOT NULL`, если домен его ещё не содержит.

Следует обратить внимание на то, что если в определении домена было указано `NOT NULL`, на уровне столбца невозможно определить допустимость использования в нем значения `NULL`. Если вы хотите чтобы на основе домена можно было определять столбцы допускающие псевдозначение `NULL` и не допускающее его, то хорошей практикой является создание домена допускающего `NULL` и указание ограничения `NOT NULL` у столбцов таблицы там где это необходимо.

9.1.5 Значение по умолчанию

Необязательное предложение `DEFAULT` определяет значение по умолчанию для столбца — это то значение, которое будет присвоено столбцу, если при добавлении новой строки в таблицу в операторе `INSERT` не указан данный столбец и его значение. Это же значение по умолчанию будет присвоено столбцу при использовании оператора `INSERT` с предложением `DEFAULT VALUES` (см. главу 11). Значение по умолчанию применяется только при выполнении оператора добавления данных `INSERT` и не оказывает никакого влияния на выполнение оператора изменения существующих в таблице данных

(UPDATE). Если в операторе изменения данных не указан какой-либо столбец, то его значение просто не изменяется.

Значением по умолчанию может быть литерал, пустое значение NULL или контекстная переменная. Литералом может быть любая самоопределенная константа соответствующего типа данных, предварительно определенный литерал или контекстная переменная SQL (подробности использования и преобразования предварительно определенных литералов и контекстных переменных см. в главе 3). Если значение по умолчанию в операторе описания столбца явно не устанавливается, то подразумевается пустое значение NULL.

Использование каких-либо выражений в значении по умолчанию недопустимо.

Пример.

Следующий столбец типа DATE имеет значением по умолчанию текущую дату на сервере (CURRENT_DATE) — дату на серверном компьютере в то время, когда выполняется данный оператор INSERT:

```
DATE_C DATE DEFAULT CURRENT_DATE
```

Если пользователь при помещении новой строки в эту таблицу не задаст значение даты, то система поместит туда текущую дату с сервера.

9.1.6 Значение NOT NULL

Необязательное предложение NOT NULL указывает, что столбцу не может быть присвоено пустое значение в операторе INSERT или UPDATE. Это предложение является обязательным для столбца, входящего в состав первичного ключа таблицы. Такое предложение для первичного ключа требуется явно указать даже в том случае, если на основании условий столбца или условий таблицы (см. далее) такому столбцу не может быть присвоено пустое значение.

Пример.

Пусть в базе данных существует справочная таблица, содержащая сведения о странах:

```
/*** Справочник стран ***/
CREATE TABLE COUNTRY (
    CODCOUNTRY CHAR(3) NOT NULL,    /* Код страны */
    NAME CHAR(30),                  /* Краткое название страны */
    FULLNAME CHAR(60),              /* Полное название страны */
    CAPITAL CHAR(15),               /* Название столицы */
    DESCR BLOB,                     /* Дополнительное описание */
    CONSTRAINT PK_COUNTRY PRIMARY KEY (CODCOUNTRY)
);
```

Здесь код страны является первичным ключом. По этой причине он объявлен с предложением NOT NULL.

9.1.7 Ограничения

Существуют четыре вида ограничений:

- первичный ключ (PRIMARY KEY);
- уникальный ключ (UNIQUE);
- внешний ключ (REFERENCES или FOREIGN KEY);
- проверочное ограничение (CHECK).

Ограничения могут быть указаны на уровне столбца («ограничения столбцов») или на уровне

таблицы («табличные ограничения»). Ограничения уровня таблицы необходимы, когда ключи (ограничение уникальности, первичный ключ или внешний ключ) должны быть сформированы по нескольким столбцам, или, когда ограничение CHECK включает несколько столбцов, т.е. действует на уровне записи. Синтаксис для некоторых типов ограничений может незначительно отличаться в зависимости от того определяется ограничение на уровне столбца или на уровне таблицы.

- Ограничение на уровне столбца указывается после определения других характеристик столбца. Оно может включать только столбец указанный в этом определении.
- Ограничения на уровне таблицы указываются после определений всех столбцов. Ограничения таблицы являются более универсальным способом записи ограничений, поскольку позволяют ограничение более чем для одного столбца таблицы.
- Вы можете смешивать ограничения столбцов и ограничения таблиц в одном операторе CREATE TABLE.

Системой автоматически создаётся индекс для первичного ключа (PRIMARY KEY), уникального ключа (UNIQUE KEY) и внешнего ключа (REFERENCES для ограничения уровня столбца, и FOREIGN KEY REFERENCES для ограничения уровня таблицы).

Именованные ограничения

Имя ограничения можно задать явно, если указать его в необязательном предложении CONSTRAINT. Рекомендуется задавать осмысленные имена ограничениям столбца. В дальнейшем при изменении характеристик таблицы к таким ограничениям будет проще обращаться по заданному имени. Имя ограничения должно быть уникальным среди имен всех ограничений столбцов и/или имен ограничений таблиц во всех таблицах базы данных, а также среди имен созданных пользователем индексов.

По умолчанию имя индекса ограничения будет тем же самым, что и самого ограничения. Если для индекса необходимо задать другое имя, то его можно указать в предложении USING.

Имена для ограничений и их индексов

Если имя ограничения не задано, то оно автоматически будет сгенерировано системой.

Ограничения уровня столбца и их индексы автоматически именуются следующим образом:

- Имена ограничений имеют следующий вид INTEG_n, где n представлено одним или несколькими числами;
- Имена индексов имеют вид RDB$PRIMARYn (для индекса первичного ключа), RDB$FOREIGNn (для индекса внешнего ключа) или RDB$n (для индекса уникального ключа), где n представлено одним или несколькими числами;

Схемы автоматического формирования имён для ограничений уровня таблицы и их индексов одинаковы.

Предложение USING

Предложение USING позволяет задать имя индекса для поддержания соответствующего ограничения первичного, уникального или внешнего ключа и указать его упорядоченность — по возрастанию значений реквизитов ключа (ASCENDING) или по убыванию их значений (DESCENDING). Если упорядоченность не задана, то предполагается ASCENDING, по возрастанию. Если индекс не указан (не задано предложение USING), то автоматически будет создан индекс с именем этого ограничения, если указано имя ограничения, или с системным именем, если не было задано имени ограничения в предложении CONSTRAINT. Для ограничения CHECK это предложение не применимо.

> Система автоматически создает индекс только для поддержания ограничений первичного, уникального и внешнего ключа. Для ограничения CHECK никакие индексы не создаются. Для этого

типа ограничений система создает соответствующие триггеры.

Ограничение UNIQUE

Ограничение `UNIQUE` определяет уникальный ключ, это означает, что при помещении в таблицу новой строки или при изменении значений существующей в таблице строки значение столбца, являющегося уникальным ключом, должно быть уникальным в таблице — в таблице не должно существовать двух разных строк, имеющих одно и то же значение такого столбца. Исключением из этого правила является только тот случай, когда уникальный ключ имеет пустое значение `NULL` (если в описании столбца не указано `NOT NULL`). Таких строк с пустым значением уникального ключа в таблице может быть произвольное количество.

Для уникального ключа система автоматически строит соответствующий индекс. Если в описании уникального ключа было указано и имя этого ограничения в предложении `CONSTRAINT`, то это имя будет присвоено индексу (если в предложении `USING` не было задано другого имени), иначе индекс получит системное имя.

В таблице может существовать произвольное количество уникальных ключей.

Ограничение уникальности может быть определено на нескольких столбцах. В этом случае вы должны определять его как ограничение уровня таблицы.

Уникальный ключ может принимать участие в связке внешний ключ/уникальный ключ для поддержания ссылочной целостности данных (предложение `REFERENCES` ограничения столбца или предложение `FOREIGN KEY` ограничения таблицы в подчиненной таблице).

Нельзя в базе данных явно создавать индекс, по структуре соответствующий уникальному ключу (см. главу 14). Такое поведение может привести к аварийному завершению работы сервера базы данных при выборке данных на основании каких-либо условий, включающих использование уникального ключа.

NULL в уникальных ключах

Согласно стандарту SQL-99 РЕД База Данных допускает одно или более значений `NULL` в столбце на который наложено ограничение `UNIQUE`. Это позволяет определить ограничение `UNIQUE` на столбцах, которые не имеют ограничения `NOT NULL`.

Для уникальных ключей, содержащих несколько столбцов, логика немного сложнее:

- Разрешено множество записей со значением `NULL` во всех столбцах ключа;
- Разрешено множество записей с различными комбинациями `null` и `not-null` значений в ключах;
- Разрешено множество записей, в которых в одном из столбцов уникального ключа содержится значение `NULL`, а остальные столбцы заполнены значениями и эти значения различны хотя бы в одном из них;

Это можно резюмировать следующим примером:

```
RECREATE TABLE t( x int, y int, z int, unique(x,y,z));
INSERT INTO t values( NULL, 1, 1 );
INSERT INTO t values( NULL, NULL, 1 );
INSERT INTO t values( NULL, NULL, NULL );
INSERT INTO t values( NULL, NULL, NULL );
INSERT INTO t values( NULL, NULL, 1 );
```

Ограничение PRIMARY KEY

Ограничение `PRIMARY KEY` определяет первичный ключ. В отличие от уникального ключа в таблице может быть только один первичный ключ.

Первичный ключ является уникальным — в таблице не может существовать двух разных строк с одним и тем же значением первичного ключа. Первичный ключ может принимать участие в связке внешний ключ/первичный ключ для поддержания ссылочной целостности данных.

Столбец, являющийся первичным ключом, должен быть описан с указанием `NOT NULL` — он не может иметь пустое значение.

Первичный ключ может состоять из одного или более столбцов таблицы.

> Первичный ключ по единственному столбцу может быть определён как на уровне столбца, так и на уровне таблицы.
>
> Первичный ключ по нескольким столбцам может быть определён только на уровне таблицы.

Для первичного ключа система также автоматически строит индекс. Если в описании первичного ключа было указано имя ограничения в предложении `CONSTRAINT` (что рекомендуется), то это имя будет присвоено индексу, если еще и в предложении `USING` не было задано другого имени.

Нельзя в базе данных создавать индекс, по структуре соответствующий первичному ключу (см. главу 14). Это может привести к аварийному завершению работы сервера базы данных при выборке данных на основании каких-либо условий, включающих использование значений первичного ключа, а также в случае, когда в плане выборки используется первичный ключ.

Использование первичных ключей

Система управления базами данных РЕД База Данных не требует обязательного присутствия в каждой таблице базы данных первичного ключа, однако наличие в таблицах первичных ключей очень желательно. Кроме того, стандарт SQL-92 требует обязательного существования для каждой таблицы первичного ключа.

Для таблицы стран, например, первичным ключом является код страны, значение которого вводит пользователь. Таблица регионов страны содержит первичный ключ, состоящий из двух столбцов — из кода страны и кода региона, что является естественным, поскольку между этими таблицами существует чисто иерархическая связь. Код страны в этом случае при добавлении новой строки в таблицу регионов может выбираться из таблицы стран, а код региона создается пользователем. Если нужна таблица, содержащая список районов региона, то для нее можно использовать первичный ключ уже из трех столбцов — код страны, код региона, код района.

Для списка людей можно в качестве первичного ключа использовать, например, номер страхового свидетельства.

Таблица, описывающая состав сотрудников какой-либо компании, может иметь первичным ключом табельный номер сотрудника. Однако такая практика не всегда срабатывает, если нужно хранить длительную историю приема и увольнения сотрудников, где возможны варианты совпадения табельных номеров сотрудников, работавших в организации в разные периоды времени.

Иными словами, варианты выбора столбцов таблицы для включения в состав первичного ключа не всегда являются очевидными. Во многих случаях хорошим решением будет создание искусственного первичного ключа. Для этого в таблицу добавляется целочисленный столбец, первичный ключ, которому при помещении в таблицу новой строки присваивается уникальное числовое значение, получаемое из объекта базы данных генератор. Для получения нового значения из генератора используется внутренняя функция `GEN_ID` или конструкция `NEXT VALUE FOR`.

Обычно для столбца искусственного первичного ключа выбирается тип данных `INTEGER`. Если таблица содержит небольшое количество записей, которые редко изменяются, то можно использовать и тип данных `SMALLINT`. Когда же в таблице присутствует очень много строк, которые к тому же часто

изменяются (одни удаляются, новые добавляются), то имеет смысл использовать даже и тип данных BIGINT. В большинстве случаев для искусственного первичного ключа все же используется тип данных INTEGER.

Пусть, например, существует таблица, хранящая данные по людям, PEOPLE. Для нее было принято решение использовать искусственный первичный ключ — целочисленный столбец PEOPLE_ID с типом данных INTEGER. Создан также генератор GEN_PEOPLE. Тогда добавить новую строку в эту таблицу можно, например, следующим образом:

```
INSERT INTO PEOPLE (PEOPLE_ID, ...)
       VALUES (GEN_ID(GEN_PEOPLE, 1), ...);
```

Для получения значения искусственного первичного ключа здесь происходит обращение к генератору. В операторе добавления новой строки при обращении к функции GEN_ID вначале текущее значение генератора увеличивается на единицу, а затем это новое значение присваивается первичному ключу новой записи.

В РЕД Базе Данных существует конструкция NEXT VALUE FOR. При ее использовании значение генератора увеличивается в точности на единицу. Эту конструкцию рекомендуется использовать вместо функции GEN_ID(). Предыдущий оператор можно записать и в следующем виде:

```
INSERT INTO PEOPLE (PEOPLE_ID, ...)
       VALUES (NEXT VALUE FOR GEN_PEOPLE, ...);
```

Ограничение FOREIGN KEY, REFERENCES

Внешний ключ должен иметь пустое значение NULL или же он должен ссылаться на первичный или уникальный ключ (родительский ключ) другой или той же самой таблицы. Понятие «ссылается» означает только лишь, что в родительской таблице должна присутствовать строка, имеющая такое же значение первичного или уникального ключа, что и внешний ключ дочерней таблицы. Ограничение внешнего ключа гарантирует, что столбец (столбцы) участник может содержать только те значения, которые существуют в указанном столбце (столбцах) родительской таблицы.

Первичный или уникальный ключ часто называют родительскими ключами.

На уровне столбца ограничение внешнего ключа определяется с использованием ключевого слова REFERENCES:

```
<ограничение столбца> ::=
   [ CONSTRAINT <имя ограничения> ]
   {   UNIQUE [<предложение USING>]
     | PRIMARY KEY [<предложение USING>]
     | REFERENCES <имя таблицы> [(<имя столбца>)] [<предложение USING>]
         [ON DELETE { NO ACTION | CASCADE | SET DEFAULT | SET NULL }]
         [ON UPDATE { NO ACTION | CASCADE | SET DEFAULT | SET NULL }]
     | CHECK (<условие столбца>)
   }
```

В предложении REFERENCES указывается имя главной, родительской таблицы и имя первичного/уникального ключа в родительской таблице, на который ссылается соответствующий ключ дочерней таблицы. Если внешний ключ ссылается на первичный, а не уникальный ключ родительской таблицы, то в предложении REFERENCES после имени родительской таблицы имя столбца первичного ключа можно не указывать, хотя это рекомендуется для документирования программной системы.

На уровне таблицы могут быть определены внешний ключ над одним или несколькими столбцами. Внешние ключи над несколькими столбцами можно определить только на уровне таблицы.

```
<ограничение таблицы> ::=
   [CONSTRAINT <имя ограничения>]
   {   UNIQUE (<столбец> [, <столбец>...]) [<предложение USING>]
     | PRIMARY KEY (<столбец> [, <столбец>...]) [<предложение USING>]
     | FOREIGN KEY (<столбец> [, <столбец>...])
     | REFERENCES <имя таблицы> [(<столбец> [, <столбец>...])] [<предложение USING>]
         [ON DELETE { NO ACTION | CASCADE | SET DEFAULT | SET NULL }]
         [ON UPDATE { NO ACTION | CASCADE | SET DEFAULT | SET NULL }]
     | CHECK (<условие столбца>)
   }
```

Синтаксис определения внешнего ключа на уровне таблицы несколько отличается. После определения всех столбцов, с их ограничения уровня столбца, вы можете определить именованное ограничение внешнего ключа уровня таблицы, используя ключевые слова `FOREIGN KEY` и имён столбцов для которых оно применяется.

Имена столбцов в главной таблице могут отличаться от тех, что указаны во внешнем ключе.

> Если целевые столбцы не указаны, то внешний ключ автоматически ссылается на столбцы первичного ключа целевой таблицы.

Действия внешнего ключа

Для обеспечения дополнительной целостности данных можно указать необязательные опции, которые обеспечат согласованность данных между родительскими и дочерними таблицами по заданным правилам:

- Предложение `ON DELETE` определяет, что произойдет с записями дочерней таблицы при удалении соответствующей строки главной таблицы. Если это предложение отсутствует, то будет установлено `RESTRICT`. Это означает, что из родительской таблицы нельзя удалить строку, для которой существуют строки в дочерней таблице, внешние ключи которых ссылаются на первичный (уникальный) ключ удаляемой строки.
- Предложение `ON UPDATE` определяет, что произойдет с записями дочерней таблицы при изменении значения первичного/уникального ключа в строке родительской таблицы. Если это предложение отсутствует, то будет установлено `RESTRICT`. Это означает, что в родительской таблице нельзя изменить значение первичного (уникального) ключа, если в дочерней таблице существуют строки, внешние ключи которых ссылаются на первичный (уникальный) ключ изменяемой строки.

Для обеспечения ссылочной целостности внешнего ключа, когда изменяется или удаляется значение связанного первичного или уникального ключа, могут быть выполнены следующие действия:

- `NO ACTION` — не будет выполнено никаких действий (значение по умолчанию). Обеспечение соответствия внешнего ключа первичному (уникальному) ключу должна выполнить сама клиентская программа, либо для этого следует написать выполняемую хранимую процедуру, к которой должно осуществляться обращение из клиентской программы в процессе удаления или изменения строки, или специально созданный пользовательский триггер до удаления (`BEFORE DELETE`) или до изменения (`BEFORE UPDATE`), выполняющий все необходимые установки значений внешнего ключа дочерней таблицы;
- `CASCADE` — при выполнении удаления (изменения) строки в главной таблице в дочерней таблице должны быть автоматически удалены (обновлены) все записи, имеющие те же значения внешнего ключа, что и значение первичного (уникального) родительского ключа удаляемой (изменяемой) строки родительской таблицы. Реализацию такого поведения выполняет автоматически созданный системный триггер. От пользователя в таком случае не требуется выполнения никаких дополнительных действий;

- SET DEFAULT — столбец внешнего ключа всех соответствующих строк в дочерней таблице устанавливается в значение по умолчанию, определенное в предложении DEFAULT этого столбца, описанного как внешний ключ. В подобной ситуации, как правило, в клиентской программе следует предпринять дополнительные меры по обеспечению непротиворечивости данных. Если значение по умолчанию для столбца внешнего ключа не задано, то столбцу присваивается значение NULL;
- SET NULL — значения внешнего ключа всех соответствующих строк в дочерней таблице устанавливаются в пустое значение NULL. Это не приведет к нарушению целостности данных, так как для внешнего ключа допустимо пустое значение.

Реализация поведения системы при удалении строки или изменении значения первичного (уникального) родительского ключа главной таблицы (за исключением задания варианта NO ACTION) осуществляется автоматически создаваемыми системными триггерами.

Для внешнего ключа система также автоматически строит индекс.

Нельзя в базе данных создавать индекс, по структуре соответствующий внешнему ключу. Это может привести к аварийному завершению работы сервера базы данных при выборке данных на основании каких-либо условий, включающих использование значений внешнего ключа.

Ограничение CHECK

Ограничение CHECK определяет условие, которому должно удовлетворять значение, помещаемое в данный столбец. Условие в предложении CHECK также иногда называется предикатом. Это логическое выражение, которое может возвращать значения TRUE (истина), FALSE (ложь) и UNKNOWN (неопределенное, неизвестное значение). Значение UNKNOWN обычно является результатом логических операций, где один из операндов имеет пустое значение NULL. Подробнее обо всех предикатах рассказано в главе 2.

Условие считается выполненным, то есть значение, помещаемое в столбец, допустимо, если предикат возвращает значение TRUE.

Такое условие может быть достаточно сложным. Это условие используется как при добавлении в таблицу новой строки (оператор INSERT), так и при изменении существующего значения столбца строки таблицы (оператор UPDATE), а также операторов, в которых может произойти одно из этих действий (UPDATE OR INSERT, MERGE). Для обеспечения выполнения условий ограничения автоматически создается системный триггер. Синтаксис условия столбца таблицы представлен в листинге 9.4.

Листинг 9.4. Синтаксис задания условия столбца таблицы

```
<условие столбца> ::= {
          <значение> <оператор сравнения> {<значение> | (<выбор одного>)}
        | <значение> [NOT] IN ({<значение> [, <значение> ...] | <поиск одного>})
        | <значение> [NOT] BETWEEN <значение> AND <значение>
        | <значение> [NOT] LIKE <шаблон> [ESCAPE '<символ>']
        | <значение> [NOT] SIMILAR TO <значение> [ESCAPE <значение>]
        | <значение> IS [NOT] NULL
        | <значение> IS [NOT] DISTINCT FROM <значение>
        | <значение> <оператор сравнения> {ALL | SOME | ANY} (<поиск одного>)
        | [NOT] EXISTS (<поиск многих>)
        | [NOT] SINGULAR (<поиск многих>)
        | <значение> [NOT] CONTAINING <значение>
        | <значение> [NOT] STARTING [WITH] <значение>
        | (<условие столбца>)
        | NOT <условие столбца>
        | <условие столбца> OR <условие столбца>
        | <условие столбца> AND <условие столбца>
}
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
<значение> ::= {
          <имя столбца> [[<элемент массива> [, <элемент массива> ...]]]
        | <литерал>
        | <контекстная переменная>
        | <выражение>
        | NEXT VALUE FOR <имя генератора>
        | GEN_ID(<имя генератора>, <значение>)
        | CAST(<значение> AS <тип данных>)
        | (<выбор одного>)
        | <обычная внутренняя функция> (<параметры>)
        | <агрегатная функция в операторе SELECT>
        | <функция UDF> [(<параметр> [, <параметр>]...)]
        | NULL }
```

Выбор одного — это оператор `SELECT`, возвращающий в точности одно значение одного столбца, получаемое из таблицы, представления или в качестве выходного параметра хранимой процедуры выбора. Пустое значение недопустимо.

Поиск одного — оператор `SELECT`, возвращающий произвольное количество значений одного столбца. Здесь возможно пустое значение.

Поиск многих — оператор `SELECT`, возвращающий ноль или произвольное количество значений нескольких столбцов.

При использовании предложения `CHECK` для столбца, базирующегося на домене, следует помнить, что выражение в `CHECK` лишь дополняет условие проверки, которое может уже быть определено в домене.

На уровне столбца или таблицы выражение в предложении `CHECK` ссылается на входящее значения с помощью с помощью идентификаторов столбцов, в отличие от доменов, где в ограничении `CHECK` для этих целей используется ключевое слово `VALUE`.

9.1.8 Нумерация столбцов

При создании таблицы происходит неявная нумерация столбцов. Первый создаваемый столбец получает номер один, следующий — номер два и т.д. Вообще говоря, порядок столбцов в таблице особого значения не имеет за исключением случая, когда в операторе добавления данных `INSERT` не задан явно список столбцов. Тем не менее, для любого столбца таблицы можно изменить номер — переместить его с одной позиции на другую. Об изменении позиции столбца см. ниже в этой главе в разделе 9.2.

9.1.9 Вычисляемые столбцы

Вычисляемый столбец задается предложением:

```
<имя столбца>  [<тип данных>] {COMPUTED [BY] | GENERATED ALWAYS AS} (<выражение>)
```

`COMPUTED [BY]` или `GENERATED ALWAYS AS` (согласно стандарту SQL-2003) эквивалентны по смыслу.

Значение такого столбца не хранится в таблице, а вычисляется, создается, при выборке данных из таблицы. Термин «вычисляемый» не обязательно означает только лишь арифметическое вычисление. Для строковых данных, например, может применяться операция конкатенации, вызов функции

получения подстроки и ряда других встроенных функций.

Выражение в этом предложении — выражение, возвращающее ровно одно значение любого типа данных, кроме BLOB или массива. Выражение может содержать любые допустимые операции, обращение к встроенным функциям и/или к функциям, определенным пользователем, UDF (см. Приложение E). Среди значений выражения допустимо использование и оператора SELECT, заключенного в круглые скобки, который при обращении к таблице (это может быть другая или та же самая таблица), представлению или хранимой процедуре выбора возвращает единственное значение или NULL. Операндами используемых в выражении операторов и функций могут быть различные константы, контекстные переменные и имена столбцов этой же таблицы. Все столбцы, используемые в выражении, должны быть определены ранее в этой таблице. Все таблицы, представления и хранимые процедуры, к которым обращаются операторы SELECT, должны уже существовать в базе данных. По этой причине вычисляемые столбцы обычно описывают в самом конце таблицы после ограничений таблицы или непосредственно перед ними. Еще один способ задания вычисляемых столбцов — добавление их в уже существующую таблицу при помощи оператора ALTER TABLE (см. далее), когда все таблицы, представления и хранимые процедуры базы данных уже описаны в системе.

Для вычисляемых полей не требуется описывать тип данных (но допустимо); вычисляемому столбцу система присваивает соответствующий тип данных, рассчитанный, исходя из вида операций и характеристик операндов в выражении вычисления. При явном указании типа столбца для вычисляемого поля результат вычисления приводится к указанному типу, то есть, например, результат числового выражения можно вывести как строку.

Пример 1. Пусть в таблице существует столбец «оклад человека», SALARY. Можно создать вычисляемый столбец с именем NET_SALARY, который будет иметь значение на 13% меньше, чем оклад (вычеты из заработной платы):

```
CREATE TABLE STAFF (
    ...
    SALARY DECIMAL(8, 2),
    NET_SALARY COMPUTED BY (SALARY * 0.87)
);
```

Вычисляемому столбцу NET_SALARY системой будет присвоен тип данных NUMERIC(18, 4). При выборке данных из этой таблицы оператором SELECT будет возвращаться и значение вычисляемого столбца, на 13 процентов меньшее, чем указанный оклад.

Пример 2. Пусть в базе данных существует таблица, описывающая различные организации, с двумя вычисляемыми столбцами, содержит код страны (см. пример выше), в которой располагается (зарегистрирована) данная организация:

```
CREATE TABLE FIRM (
    COD INTEGER NOT NULL,
    NAME1 CHAR(50),
    CODCOUNTRY CHAR(3),
    COUNTRYNAME COMPUTED BY ((SELECT NAME
                              FROM COUNTRY
                              WHERE COUNTRY.CODCOUNTRY = FIRM.CODCOUNTRY)),
    FULLCOUNTRYNAME COMPUTED BY ((SELECT FULLNAME
                                  FROM COUNTRY
                                  WHERE COUNTRY.CODCOUNTRY = FIRM.CODCOUNTRY))
);
```

В этой таблице присутствует два вычисляемых столбца. Один получит тип данных VARCHAR(30), поскольку отыскиваемый при использовании оператора SELECT столбец из справочной таблицы (краткое название страны) имеет тип данных VARCHAR(30), другой вычисляемый столбец, отыскиваемый также в таблице стран, получает тип данных VARCHAR(60). Обратите внимание, что оператор SELECT заключен в двойную пару круглых скобок. Во всех синтаксических конструкциях, где присутствует

одиночный оператор SELECT (оператор, возвращающий ровно одно значение одного столбца или пустое значение NULL), этот оператор должен быть заключен в круглые скобки. Внешняя пара скобок требуется, потому что выражение для любого вычисляемого столбца по правилам синтаксиса также должно заключаться в круглые скобки.

В обоих операторах SELECT в предложениях WHERE именам столбцов предшествует имя соответствующей таблицы и точка. Это так называемые уточненные имена. Имя таблицы здесь требуется, чтобы устранить возникающую неопределенность, поскольку столбец с именем CODCOUNTRY присутствует в обеих таблицах — и в FIRM, и в COUNTRY. Для уточненных имен возможно использование и псевдонимов (или алиасов, alias) таблиц. Использование псевдонимов может несколько сократить количество символов, набираемых для выполнения оператора, однако их применение имеет больший смысл, когда в сложном запросе одна и та же таблица встречается в нескольких различных конструкциях оператора SELECT. Если для таблицы задан псевдоним, то во всех уточненных именах столбцов можно использовать только псевдонимы, использование имени таблицы в этом случае недопустимо. При отсутствии псевдонима используется имя таблицы. Для главной таблицы, таблицы самого верхнего уровня, используемой в первом операторе SELECT, уточняющее имя можно не указывать.

Например, последний вычисляемый столбец этого же примера мог бы быть записан в следующем виде:

```
FULLCOUNTRYNAME COMPUTED BY ((SELECT FULLNAME
                              FROM COUNTRY C
                              WHERE C.CODCOUNTRY = FIRM.CODCOUNTRY))
```

Здесь для таблицы COUNTRY задается псевдоним C. После этого в любом предложении данного оператора обращаться к данной таблице можно только по псевдониму, а не по имени таблицы.

Поскольку таблица COUNTRY является главной таблицей в операторе SELECT, то ее имя или псевдоним можно в операторе не указывать. Последнее определение вычисляемого столбца без каких-либо ошибок можно записать и в следующем виде:

```
FULLCOUNTRYNAME COMPUTED BY ((SELECT FULLNAME
                              FROM COUNTRY
                              WHERE CODCOUNTRY = FIRM.CODCOUNTRY))
```

Для таблицы же FIRM псевдоним или имя таблицы (в данном случае, именно имя этой таблицы) обязательно должно быть указано.

Подробнее о связи псевдонимов и имен таблиц см. в разделе 11.1.

Для того чтобы иметь возможность просматривать все данные из приведенной в предыдущем примере таблицы FIRM пользователь, соединенный с базой данных, должен иметь привилегии просмотра не только к этой таблице, но и к справочной таблице COUNTRY. Если же производится выборка из таблицы (таблиц), получаемых при обращении к хранимой процедуре выбора, то пользователь должен иметь соответствующие полномочия к этой хранимой процедуре. Однако если пользователь выполняет оператор SELECT, который выбирает данные только из таблицы FIRM, а в заданном списке выбора отсутствуют столбцы COUNTRYNAME и FULLCOUNTRYNAME из таблицы стран, то пользователю нет необходимости иметь полномочия к таблице COUNTRY.

Использование возможностей оператора SELECT см. в разделе 11.1. Описание полномочий пользователя к таблицам, процедурам, триггерам и представлениям см. в документе «Руководство администратора».

9.1.10 Столбцы идентификации

Столбцы идентификации могут быть определены с помощью предложения `GENERATED BY DEFAULT AS IDENTITY`, либо предложения `GENERATED ALWAYS AS IDENTITY`. Столбец идентификации представляет собой столбец, связанный с внутренним генератором последовательностей.

Если столбец идентификации задан как `GENERATED BY DEFAULT`, то его значение будет увеличиваться и использовано как значение по умолчанию при каждой вставке, только в том случае, если значение этого столбца не задано явно.

Чтобы использовать сгенерированное по умолчанию значение, необходимо либо указать ключевое слово `DEFAULT` при вставке в столбец идентификации, или просто не упоминать столбец идентификации в списке столбцов для вставки. В противном случае будет использовано указанное вами значение.

Если столбец идентификации задан как `GENERATED ALWAYS`, то его значение будет увеличиваться при каждой вставке. При попытке явно присвоить значение столбца идентификации в операторе `INSERT`, будет выдано сообщение об ошибке. В операторе `INSERT` вы можете указать ключевое слово `DEFAULT` вместо значения для столбца идентификации.

```
create table greetings (id INT GENERATED ALWAYS AS IDENTITY, name CHAR(50));
INSERT INTO greetings VALUES (DEFAULT, 'hello');
INSERT INTO greetings(name) VALUES ('bonjour');
INSERT INTO greetings(id, name) VALUES (10, 'hello');         -- Запрещено
```

Необязательное предложение `START WITH` позволяет указать начальное значение отличное от нуля. Предложение `INCREMENT [BY]` устанавливает значение приращения. Значение приращения должно быть отлично от 0. По умолчанию значение приращения равно 1.

Идентификационные столбцы неявно являются `NOT NULL` столбцами.

Тип данных столбца идентификации должен быть целым числом с нулевым масштабом. Допустимыми типами являются `SMALLINT, INTEGER, BIGINT, NUMERIC(x,0)` и `DECIMAL(x,0)`.

Идентификационный столбец не может иметь значений по умолчанию и вычисляемых значений.

Идентификационный столбец не может быть изменён в обычный столбец. И наоборот.

9.1.11 Привилегии выполнения

Необязательное предложение `SQL SECURITY {DEFINER | INVOKER}` определяет, в контексте какого пользователя будут вычисляться вычисляемые столбцы. Ключевое слово `INVOKER` (значение по умолчанию) указывает, что вычисляемые столбцы вычисляются с правами текущего пользователя. Задание ключевого слова `DEFINER` означает, что вычисляемые столбцы вычисляются с правами к объектам базы данных владельца (создателя) таблицы.

Значение по умолчанию на уровне всей базы данных можно изменить оператором `ALTER DATABASE SET DEFAULT SQL SECURITY`.

9.2 Изменение таблиц

Описание существующих в базе данных таблиц (характеристик столбцов, их порядка, наличие различных ключей или ограничения `CHECK`) можно изменять после создания таблиц.

Для изменения структуры существующих таблиц используется оператор `ALTER TABLE`.

Листинг 9.5. Синтаксис оператора изменения таблицы `ALTER TABLE`

```
ALTER TABLE <имя таблицы> <операция изменения> [, <операция изменения>...];

<операция изменения> ::= {
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
      ADD <определение столбца>
    | ADD <ограничение таблицы>
    | DROP <имя столбца>
    | DROP CONSTRAINT <ограничение столбца или таблицы>
    | ALTER [COLUMN] <имя столбца> <модификация столбца>
    | ALTER SQL SECURITY {DEFINER|INVOKER}
    | DROP SQL SECURITY
    | {ENABLE | DISABLE} PUBLICATION
    | SET TABLESPACE [TO] {<имя табличного пространства> | PRIMARY}
}

<определение столбца> ::= <опр-е обычного столбца>
                        | <опр-е вычисляемого столбца>
                        | <опр-е идентификационного столбца>

<определение обычного столбца> ::=
             <имя столбца> { <тип данных> | <имя домена>}
             [DEFAULT {<литерал> | NULL | <контекстная переменная>}]
             [NOT NULL]
             [<ограничение столбца>]
             [COLLATE <порядок сортировки>]

<определение вычисляемого столбца> ::=
             <имя столбца> [<тип данных>]
            {COMPUTED [BY] | GENERATED ALWAYS AS} (<выражение>)

<определение идентификационного столбца> ::=
       <имя столбца> [<тип данных>]
      {ALWAYS|GENERATED BY} DEFAULT AS IDENTITY [(START WITH <начальное значение>)]
      [<ограничение столбца>]

<модификация столбца> ::= TO <новое имя столбца>
                        | POSITION <новая позиция>
                        | <мод-я обычного столбца>
                        | <мод-я вычисляемого столбца>
                        | <мод-я идентификационного столбца>

<модификация обычного столбца> ::=
             TYPE { <тип данных> | <имя домена> }
           | SET DEFAULT { <литерал> | NULL | <контекстная переменная>}
           | DROP DEFAULT
           | SET NOT NULL
           | DROP NOT NULL

<модификация вычисляемого столбца> ::=
   [TYPE <тип данных>] {GENERATED ALWAYS AS | COMPUTED [BY]} (<выражение>)

<модификация идентификационного столбца> ::=
             <опции автоинкремента>
           | SET GENERATED {ALWAYS|BY DEFAULT} [ <опции автоинкремента> ...]
           | DROP IDENTITY
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
<опции автоинкремента> ::= RESTART [ WITH <стартовое значение> ]
                         | SET INCREMENT [BY] <приращение>

<ограничение столбца> ::=
   [CONSTRAINT <имя ограничения>]
   { UNIQUE [<предложение USING>] [[IN] TABLESPACE {<имя табл. пространства>|PRIMARY}]
   | PRIMARY KEY [<предложение USING>] [[IN] TABLESPACE {<имя табл. прос-ва>|PRIMARY}]
   | REFERENCES <имя таблицы> [(<имя столбца>)]
         [ON DELETE { NO ACTION | CASCADE | SET DEFAULT | SET NULL }]
         [ON UPDATE { NO ACTION | CASCADE | SET DEFAULT | SET NULL }]
         [<предложение USING>] [[IN] TABLESPACE { <имя табл. пространства> | PRIMARY}]
   | CHECK (<условие столбца>)
   }

<ограничение таблицы> ::=
   [CONSTRAINT <имя ограничения>]
   { UNIQUE (<столбец> [, <столбец>...]) [<предложение USING>]
       [[IN] TABLESPACE { <имя табличного пространства> | PRIMARY}]
   | PRIMARY KEY (<столбец> [, <столбец>...]) [<предложение USING>]
       [[IN] TABLESPACE { <имя табличного пространства> | PRIMARY}]
   | FOREIGN KEY (<столбец> [, <столбец>...])
   | REFERENCES <имя таблицы> [(<столбец> [, <столбец>...])]
       [ON DELETE { NO ACTION | CASCADE | SET DEFAULT | SET NULL }]
       [ON UPDATE { NO ACTION | CASCADE | SET DEFAULT | SET NULL }]
       [<предложение USING>] [[IN] TABLESPACE { <имя табл. пространства> | PRIMARY}]
   | CHECK (<условие столбца>)
   }

<предложение USING> ::= USING [ASC[ENDING] | DESC[ENDING]] INDEX <имя индекса>
```

Изменять таблицу может ее владелец, администратор и пользователь с ролью `ALTER ANY TABLE`.

В одном операторе можно выполнить произвольное количество изменений в таблице. Различные операции по изменению существующей таблицы отделяются друг от друга запятыми.

Некоторые изменения структуры таблицы увеличивают счётчик форматов, закреплённый за каждой таблицей. Количество форматов для каждой таблицы ограничено значением 255 или 32000 для каждого представления. После того, как счётчик форматов достигнет этого значения, вы не сможете больше менять структуру таблицы или представления без сброса счётчика.

Для сброса счётчика форматов необходимо сделать резервное копирование и восстановление базы данных.

9.2.1 Добавление нового столбца

Для добавления нового столбца в существующую в базе данных таблицу следует ввести в операторе
`ALTER TABLE` следующую конструкцию:

```
ADD <определение столбца>
```

В определении добавляемого столбца присутствует как описание характеристик столбца, так и возможное ограничение столбца. В операторе можно задать значение по умолчанию для столбца (пред-

ложение DEFAULT), задать ограничение недопустимости пустого значения NOT NULL, добавить различные ограничения столбца или установить порядок сортировки (предложение COLLATE).

Синтаксис определения столбца полностью совпадают с синтаксисом, описанным в операторе CREATE TABLE.

Если в таблице, куда добавляется новый столбец, уже существуют строки с данными, то к каждой строке присоединяется новый столбец с пустым значением NULL. На это не влияет наличие в описании добавляемого столбца предложения DEFAULT.

Добавление столбца с ограничением NOT NULL без значения DEFAULT будет неудачным, если в таблице есть строки с данными. При добавлении столбца с ограничением NOT NULL рекомендуется либо установить для него значение по умолчанию, либо создать его как ненулевой, обновить значения столбца в существующих строках, а затем добавить ограничение NOT NULL.

Когда добавляется новое условие CHECK, существующие данные не проверяются на соответствие. Рекомендуется предварительно проверить существующие данные на соответствие новому условию CHECK.

Добавление столбца идентификации поддерживается, но оно будет успешным только в том случае, если таблица пуста. Добавление столбца идентификации будет неудачным, если в таблице есть хотя бы одна строка.

При каждом добавлении нового столбца номер формата увеличивается на единицу.

9.2.2 Добавление ограничения таблицы

Для добавления нового ограничения таблицы нужно в операторе ALTER TABLE использовать следующее предложение:

```
ADD <ограничение таблицы>
```

Таким способом нельзя добавить ограничение столбца, можно добавлять только ограничения для всей таблицы. Это не является серьезным недостатком, поскольку ограничение таблицы может относиться и к одному единственному столбцу.

Синтаксис добавления ограничения таблицы полностью совпадают с синтаксисом, описанным в операторе CREATE TABLE.

Добавление нового ограничения таблицы не влечёт за собой увеличение номера формата.

Во многих случаях бывает удобным добавлять ограничения таблиц после создания всех таблиц базы данных. В первую очередь это касается ограничения внешнего ключа. При определении ограничения внешнего ключа родительская таблица, на первичный или уникальный ключ которой ссылается внешний ключ, уже должна существовать в базе данных. Если внешний ключ ссылается именно на первичный ключ родительской таблицы, то список столбцов, входящих в состав этого первичного ключа, можно в операторе не указывать.

Пример 1. Пусть существует таблица COUNTRY, которая содержит сведения о странах. Ее первичным ключом является столбец CODCOUNTRY. Таблица регионов REGION также содержит столбец CODCOUNTRY, который должен быть внешним ключом, ссылающимся на таблицу стран. Для добавления ограничения внешнего ключа в таблицу регионов после создания всех таблиц базы данных нужно ввести и выполнить оператор:

```
ALTER TABLE REGION
    ADD CONSTRAINT FK_REGION
       FOREIGN KEY (CODCOUNTRY)
       REFERENCES COUNTRY (CODCOUNTRY)
          ON DELETE CASCADE
          ON UPDATE CASCADE;
```

В данном примере имена столбцов внешнего ключа дочерней таблицы и первичного ключа родительской таблицы совпадают. На самом деле это не является требованием системы. Должны совпадать только типы данных внешнего и первичного (уникального) ключей и размерность строковых столбцов. В этом примере можно было вообще не указывать столбец первичного ключа родительской таблицы, потому что именно первичный ключ в родительской таблице предполагается по умолчанию в описании внешнего ключа.

Пример 2. Добавление проверочного ограничения и внешнего ключа (таблица `JOB` содержится в базе данных `employee.fdb`.):

```
ALTER TABLE JOB
    ADD CONSTRAINT CHK_SALARY CHECK (MIN_SALARY < MAX_SALARY),
    ADD FOREIGN KEY (JOB_COUNTRY)
       REFERENCES COUNTRY (CODCOUNTRY)
          ON UPDATE CASCADE
          ON DELETE SET NULL;
```

При добавлении нового ограничения `CHECK` не осуществляется проверка соответствия ему ранее внесённых данных. Поэтому перед добавлением такого ограничения рекомендуем производить предварительную проверку данных в таблице.

9.2.3 Удаление столбца таблицы

Для удаления существующего столбца таблицы в операторе изменения таблицы надо ввести:

```
DROP <имя столбца>;
```

Операция удаления столбцов требует определенной осторожности. Прежде чем удалять столбец, нужно удалить все зависимости в базе данных, связанные с этим столбцом. Иными словами, нужно удалить все ссылки на этот столбец. Такие ссылки могут присутствовать:

- *в ограничениях столбцов или таблицы*; такие ограничения могут существовать как в текущей, корректируемой, таблице, так и в любой другой существующей таблице базы данных. В первую очередь это могут быть ограничения первичного, уникального или внешнего ключа, в состав которого входит удаляемый столбец. Затем это могут быть ограничения внешних ключей в других таблицах, которые ссылаются на первичный или уникальный ключ корректируемой таблицы, если удаляемый столбец входит в состав первичного (уникального) ключа. Наконец, это могут быть ограничения `CHECK` данной или иных таблиц, в условиях которых присутствует удаляемый столбец;
- *в индексах*, когда удаляемый столбец входит в состав каких-либо индексов базы данных, созданных пользователем для изменяемой таблицы;
- *в хранимых процедурах, функциях и триггерах*, где присутствуют обращения к значениям удаляемого столбца;
- *в представлениях*, где удаляемый столбец может присутствовать в списке выбора, а также в предложении `ON` соединяемых таблиц, определяющем условие соединения, в предложении `WHERE`, определяющем условие выборки, или в предложениях `ORDER BY`, существующих в представлениях, задающих упорядоченность результата выборки данных.

При каждом удалении столбца номер формата увеличивается на единицу.

9.2.4 Удаление ограничения

Чтобы удалить существующее ограничение столбца или ограничение таблицы следует в операторе изменения таблицы ввести:

```
DROP CONSTRAINT <имя ограничения столбца или таблицы>;
```

Для удобства выполнения такой операции желательно явно именовать все ограничения столбцов и таблиц базы данных при создании этих ограничений. Иначе придется просматривать записи системной таблицы RDB$RELATION_CONSTRAINTS, чтобы определить необходимое имя.

Ограничение первичного ключа или уникального ключа не могут быть удалены, если на него ссылается ограничение FOREIGN KEY в другой таблице. В этом случае, необходимо удалить ограничение FOREIGN KEY до удаления PRIMARY KEY или UNIQUE ключа, на которые оно ссылается. Другие ограничения — ограничения внешнего ключа и ограничения CHECK не имеют никаких зависимостей, делающих невозможным их удаление.

Удаление ограничения столбца или ограничения таблицы не влечёт за собой увеличение номера формата.

9.2.5 Изменение существующего столбца

При использовании оператора ALTER TABLE есть несколько вариантов изменения характеристик существующего столбца таблицы с помощью предложения ALTER [COLUMN]:

- изменение имени (не изменяет номер формата);
- изменение типа данных (увеличивает номер формата на единицу);
- изменение позиции столбца в списке столбцов таблицы (не изменяет номер формата);
- удаление значения по умолчанию столбца (не изменяет номер формата);
- добавление значения по умолчанию столбца (не изменяет номер формата);
- изменение типа и выражения для вычисляемого столбца (не изменяет номер формата);
- изменение столбцов идентификации;
- добавление ограничения NOT NULL (не изменяет номера формата);
- удаление ограничения NOT NULL (не изменяет номера формата);
- изменение типа идентификационного столбца или переопределение его в обычный;
- перезапуск идентификационного столбца;
- изменение приращения идентификационного столбца.

Изменение имени

Для изменения имени столбца в операторе изменения таблицы ALTER TABLE используется конструкция:

```
ALTER [COLUMN] <имя столбца> TO <новое имя столбца>
```

Ключевое слово TO переименовывает существующий столбец. Новое имя столбца должно быть уникальным в рамках таблицы.

Невозможно изменение имени столбца, если этот столбец включен в какое-либо ограничение - первичный или уникальный ключ, внешний ключ, ограничение столбца или ограничение таблицы CHECK.

Имя столбца также нельзя изменить, если этот столбец таблицы используется в каком-либо триггере, в хранимой процедуре или в представлении, созданном пользователем. Если для таблицы был автоматически создан триггер для поддержания какого-либо ограничения, то соответствующее имя в триггере будет автоматически изменено при изменении имени столбца.

```
ALTER TABLE STOCK
ALTER COLUMN MODELNAME TO NAME;
```

Изменение типа данных столбца

Для изменения типа данных столбца используется следующая синтаксическая конструкция в операторе изменения таблицы ALTER TABLE:

```
ALTER [COLUMN] <имя столбца> TYPE <новый тип данных>
```

Ключевое слово TYPE изменяет тип данных существующего столбца на другой, допустимый тип. Изменение типа, которое может привести к потере данных, не допускается. Например, количество символов в новом типе для столбца CHAR или VARCHAR не может быть меньше, чем в исходном типе.

Если столбец был объявлен как массив, то изменить ни его тип, ни размерность нельзя.

Тип BLOB можно изменить только на BLOB такого же подтипа.

Нельзя изменить тип данных столбца, который задействован в ограничениях FOREIGN KEY, PRIMARY KEY и UNIQUE.

```
ALTER TABLE STOCK
ALTER COLUMN ITEMID TYPE BIGINT;
```

Изменение позиции столбца

Для изменения позиции столбца в таблице используется следующая конструкция в операторе изменения таблицы:

```
ALTER [COLUMN] <имя столбца> POSITION <номер позиции>
```

Это самая простая операция по изменению таблицы, почти не приводящая к неприятным последствиям. Ошибки могут возникнуть лишь в том случае, если существуют операторы INSERT, в которых явно не указан список имен добавляемых столбцов. При изменении позиции столбца такие операторы станут работать неправильно и могут вызвать исключения базы данных. Это также может привести к неверному выполнению оператора SELECT, если в списке выбора был указан выбор всех столбцов таблицы (символ *), а в предложении ORDER BY был задан номер столбца, по которому выполняется упорядочивание полученных данных.

Нумерация позиций столбцов начинается с 1. Если указана позиция меньше 1, будет возвращено сообщение об ошибке. Если номер указанной позиции больше, чем количество столбцов в таблице, то новая позиция будет скорректирована, чтобы соответствовать количеству столбцов.

```
ALTER TABLE STOCK
ALTER COLUMN ITEMID POSITION 5;
```

Удаление значения по умолчанию столбца

Для удаления значения по умолчанию столбца используется следующая конструкция:

```
ALTER [COLUMN] <имя столбца> DROP DEFAULT
```

Если столбец основан на домене со значением по умолчанию, значение по умолчанию для столбца будет возвращено к значению по умолчанию, указанному для домена.

Если удаление значения по умолчанию производится над столбцом, у которого нет значения по умолчанию, или чьё значение по умолчанию основано на домене, то это приведёт к ошибке выполнения данного оператора.

```
ALTER TABLE STOCK
ALTER COLUMN MODEL DROP DEFAULT;
```

Добавление значения по умолчанию столбца

Для добавления значения по умолчанию столбца используется конструкция:

```
ALTER [COLUMN] <имя столбца> SET DEFAULT <ограничение столбца>
```

Если столбец уже имел значение по умолчанию, то оно будет заменено новым. Значение по умолчанию, указанное для столбца, всегда отменяет значение, унаследованное от домена.

```
ALTER TABLE STOCK
ALTER COLUMN MODEL SET DEFAULT 'RTGZ';
```

Добавление ограничения NOT NULL

Предложение `SET NOT NULL` добавляет ограничение `NOT NULL` для столбца таблицы.

```
ALTER [COLUMN] <имя столбца>  SET NOT NULL
```

Добавление ограничения `NOT NULL` сопровождается полной проверкой данных в таблице, поэтому перед изменением убедитесь, что в столбце нет значений `NULL`.

Явное ограничение `NOT NULL` на столбце, основанном на домене, преобладает над параметрами домена. В этом случае снятие ограничения `NOT NULL` с домена не распространяется на столбец таблицы.

```
ALTER TABLE STOCK
ALTER COLUMN ITEMID SET NOT NULL;
```

Удаление ограничения NOT NULL

Предложение `DROP NOT NULL` удаляет ограничение `NOT NULL` для столбца таблицы:

```
ALTER [COLUMN] <имя столбца>  DROP NOT NULL
```

Снятие ограничения `NOT NULL` со столбца, основанного на домене, который также имеет ограничение `NOT NULL`, не будет иметь эффекта, пока ограничение `NOT NULL` не будет снято и с домена.

```
ALTER TABLE STOCK
ALTER COLUMN ITEMID DROP NOT NULL;
```

Изменение вычисляемых столбцов

Тип данных и выражение, на котором основан вычисляемый столбец, можно изменить с помощью предложений `COMPUTED [BY]` и `GENERATED ALWAYS AS`:

```
ALTER [COLUMN] <имя столбца> [TYPE  <тип данных>]
  { GENERATED ALWAYS AS | COMPUTED [BY] } (<выражение>)
```

Преобразование обычного столбца в вычисляемый и наоборот не допускается.

Изменение столбцов идентификации

Для столбцов идентификации позволено изменять способ генерации, начальное значение и значение приращения.

Изменение способа генерации столбца идентификации

Предложение `SET GENERATED {ALWAYS | BY DEFAULT}` позволяет изменить способ генерации столбца идентификации. Нельзя использовать этот параметр для изменения обычного столбца на столбец идентификации.

```
ALTER TABLE objects
ALTER ID SET GENERATED ALWAYS;
```

Существует два способа генерации столбца идентификации:

- `BY DEFAULT` столбцы позволяют переписать сгенерированное системой значение в операторах `INSERT, UPDATE OR INSERT, MERGE` просто указав значение этого столбца в списке значений.
- `ALWAYS` столбцы не позволяют переписать сгенерированное системой значение, при попытке переписать значение такого столбца идентификации будет выдана ошибка. Переписать значение этого столбца в операторе `INSERT` можно только при указании директивы `OVERRIDING SYSTEM VALUE`.

Изменение начального значения столбца идентификации

Предложение `RESTART` перезапускает последовательность, используемую для генерации значений столбца идентификации. Если указано только предложение `RESTART`, то последовательность сбрасывается на начальное значение, указанное при определении столбца идентификации. Если указано необязательное условие `WITH`, последовательность перезапускается с указанного значения.

Последующие перезапуски (без `WITH`) будут использовать значение `START WITH`, указанное при определении столбца идентификации. В настоящее время изменить начальное значение невозможно.

```
ALTER TABLE objects
ALTER ID RESTART WITH 100;
```

Изменение приращения столбца идентификации

Предложение `SET INCREMENT [BY]` позволяет изменить значение приращения столбца идентификации. Значение приращения должно быть отлично от 0.

```
ALTER TABLE objects
ALTER ID SET INCREMENT BY 2;
```

В одном операторе можно изменить сразу несколько свойств столбца идентификации, например:

```
ALTER TABLE objects
ALTER ID SET GENERATED ALWAYS RESTART SET INCREMENT BY 2;
```

Преобразование столбца идентификации в обычный

Предложение DROP IDENTITY удаляет связанную со столбцом идентификации системную

```
ALTER TABLE objects
ALTER ID DROP IDENTITY;
```

9.2.6 Изменение прав на работу с таблицей

Необязательное предложение ALTER SQL SECURITY {DEFINER|INVOKER} определяет, в контексте какого пользователя будет проходить работа с таблицей. Ключевое слово INVOKER (значение по умолчанию) указывает, что вычисляемые столбцы вычисляются с привилегиями вызывающего пользователя. Задание ключевого слова DEFINER означает, что вычисляемые столбцы вычисляются с привилегиями определяющего пользователя (владельца).

```
ALTER TABLE COUNTRY
ALTER SQL SECURITY DEFINER;
```

Если SQL SECURITY изменяется для таблицы, то триггеры, для которых явно не указано свойство SQL SECURITY, не узнают об изменении до следующей загрузки триггера в кэш метаданных. Кроме того триггеры наследуют привилегии выполнения у таблицы, если они не переопределены у самих триггеров.

Предложение DROP SQL SECURITY удаляет опцию, указанную при создании. После удаления применяется значение базы данных по умолчанию.

Значение по умолчанию на уровне всей базы данных можно изменить оператором ALTER DATABASE SET DEFAULT SQL SECURITY.

9.2.7 Перемещение таблицы в табличное пространство

Для таблицы может быть указано табличное пространство для отдельного физического хранения. По умолчанию все индексы таблицы создаются в том же табличном пространстве, что и сама таблица.

```
ALTER TABLE <имя таблицы> SET TABLESPACE <имя табличного пространства>;
```

Для перемещения таблицы в основной файл базы данных примените оператор:

```
ALTER TABLE <имя таблицы> SET TABLESPACE TO PRIMARY
```

Операторы перемещения таблицы в табличное пространство требуют наличия единственного подключения к базе данных. Это временное ограничение, что делает процедуру перемещения более надежной

9.3 Удаление таблиц

Для удаления существующей таблицы используется оператор DROP TABLE.

Листинг 9.6. Синтаксис оператора удаления таблицы DROP TABLE

```
DROP TABLE <имя таблицы>;
```

Удалять таблицу может ее владелец, администратор и пользователь с привилегией DROP ANY TABLE.

Нельзя удалить таблицу, которая является родительской в связке внешний ключ/первичный (уникальный) ключ. Нельзя также удалить таблицу, на которую существуют ссылки в триггерах (за исключением триггеров, написанных пользователем именно для этой таблицы), и таблицу, которая используется в хранимой процедуре или в представлении.

Таблица, используемая в какой-либо активной транзакции, не будет удалена до завершения (подтверждения или отмены) этой транзакции.

При успешном удалении таблицы автоматически будут удалены все ее данные, триггеры, созданные для этой таблицы (пользователем или автоматически системой), а также все индексы, построенные автоматически системой управления базами данных или пользователем для такой таблицы.

9.4 Пересоздание таблицы

Таблица пересоздается оператором RECREATE TABLE.

Листинг 9.7. Синтаксис оператора пересоздания таблицы RECREATE TABLE

```
RECREATE [GLOBAL TEMPORARY] TABLE <имя таблицы>
   [EXTERNAL [FILE] '<спецификация файла>']
   (<определение столбца> [, { <определение столбца> | <ограничение таблицы>}...])
   [ON COMMIT {DELETE | PRESERVE} ROWS]
   [SQL SECURITY {DEFINER | INVOKER}];
```

Этот оператор создаёт или пересоздает таблицу. Если таблица с таким именем уже существует, то оператор RECREATE TABLE попытается удалить её и создать новую. Оператор RECREATE TABLE не выполнится, если существующая таблица имеет зависимости.

Данная операция доступна и для глобальных временных таблиц (GTTs) с синтаксисом, аналогичным оператору CREATE GLOBAL TEMPORARY TABLE.

Полное описание определений столбцов и ограничений таблицы смотрите в разделе CREATE TABLE.

9.5 Примечание к таблице и ее столбцам

К существующей таблице и к каждому ее столбцу можно создавать примечания при помощи оператора COMMENT. Это хорошее средство документирования разрабатываемой базы данных, помимо тех комментариев, которые присутствуют в скриптах, где создаются все необходимые таблицы и другие объекты базы данных.

Для создания примечания к таблице используется следующий синтаксис (см. листинг 9.8):

Листинг 9.8. Синтаксис оператора создания примечания таблицы

```
COMMENT ON TABLE <имя таблицы> IS {'<текст>' | NULL};
```

Можно указать или текст примечания или задать NULL. В последнем случае будет удалено существующее примечание, если оно было ранее создано.

Например, чтобы создать примечание для таблицы PEOPLE, нужно выполнить оператор:

```
COMMENT ON TABLE PEOPLE IS 'Таблица, содержащая сведения о людях';
```

Чтобы создать примечание к столбцу уже созданной таблицы, используется следующий синтаксис:

```
COMMENT ON COLUMN <имя таблицы>.<столбец> IS {'<текст>' | NULL};
```

Например, чтобы создать примечание для столбца `LAST_NAME` таблицы `PEOPLE`, нужно выполнить:

```
COMMENT ON COLUMN PEOPLE.LAST_NAME IS 'Фамилия человека';
```

Глава 10

Табличное пространство (TABLESPACE)

Табличные пространства используются, прежде всего, для разделения и хранения определенных объектов базы данных — таких как индексы и таблицы. Табличные пространства не имеют отношения к логической структуре базы данных, а предназначены для указания места хранения данных на физических носителях. Различные объекты одной базы данных, например, индекс и таблица, могут физически храниться в разных пространствах.

Ниже приведен краткий список преимуществ использования табличных пространств:

- Позволяют расширить текущие лимиты размера базы данных, даже для небольших размеров таблицы.
- Дает возможность контролировать использование базой данных доступного места и оптимизировать быстродействие. Например, пространство, используемое для индексов, можно разместить на быстрых накопителях, а неактивную или архивную части базы данных перемещать на медленные носители большого объема.

Для создания табличного пространства понадобится оператор `CREATE TABLESPACE`:

Листинг 10.1. Синтаксис оператора создания табличного пространства `CREATE TABLESPACE`

```
CREATE TABLESPACE <имя табличного пространства> FILE '<путь к файлу>';
```

Права на создание табличных пространств есть только у администраторов и пользователей с привилегией `CREATE TABLESPACE`. Максимально доступно 254 табличных пространства.

Для создания табличного пространства необходимо указать путь к файлу. Можно указывать как абсолютный путь, так и относительный, в том числе с использованием псевдонима директории, заданного в `directories.conf` в секции `tablespaces`. В `directories.conf` указывается реальный путь к директории с псевдонимом `<псевдоним директории>`. Путь к файлу табличного пространства с использованием псевдонима директории указывается в формате: `<псевдоним директории>/<файл>`, например, `dir/file.dat`. Первый компонент пути (`dir`) будет обработан как псевдоним директории. Если псевдоним не будет найден в `directories.conf`, то путь будет обработан как относительный (относительно директории, в которой размещен основной файл базы данных). Все директории, используемые в пути, должны быть созданы заранее. В системную таблицу `RDB$TABLESPACES` в любом случае будет записан абсолютный путь к файлу табличного пространства.

Пример создания табличного пространства с использованием псевдонима директории:

```
CREATE TABLESPACE TS1 FILE 'dir/ts1.dat';
```

После создания табличного пространства вы можете перемещать таблицы или индексы в него. Чтобы создать таблицу в табличном пространстве, воспользуйтесь оператором:

```
CREATE TABLE <имя таблицы> (<столбец> [, <столбец>...])
    [IN] TABLESPACE {<имя табличного пространства> | PRIMARY};
```

Чтобы переместить существующую таблицу в табличное пространство, примените оператор:

```
ALTER TABLE <имя таблицы> SET TABLESPACE [TO] <имя табл. пространства>;
```

Ключевое слово `PRIMARY` можно использовать в качестве имени табличного пространства, если необходимо ссылаться на основной файл базы данных.

Для перемещения таблицы в основной файл базы данных примените оператор:

```
ALTER TABLE <имя таблицы> SET TABLESPACE [TO] PRIMARY;
```

Операторы перемещения таблицы в табличное пространство требуют наличия единственного подключения к базе данных. Это временное ограничение, что делает процедуру перемещения более надежной.

По умолчанию индекс будет создан в том табличном пространстве, в котором располагается таблица, к которой он относится. Если вы хотели бы создать его в своем tablespace, примените оператор:

```
CREATE INDEX <имя индекса> [IN] TABLESPACE {<имя табличного пространства> | PRIMARY};
```

Чтобы переместить существующий индекс в табличное пространство, примените оператор:

```
ALTER INDEX <имя индекса> SET TABLESPACE [TO] <имя табл. пространства> ;
```

Для перемещения индекса в основной файл базы данных примените оператор:

```
ALTER INDEX <имя индекса> SET TABLESPACE [TO] PRIMARY;
```

Операторы перемещения индекса в табличное пространство требуют наличия единственного подключения к базе данных. Это временное ограничение, что делает процедуру перемещения более надежной.

Можно указать табличное пространство для ограничений PK, FK, UNIQUE, созданных в CREATE TABLE и ALTER TABLE. Синтаксис описан в главе 9.

Для изменения путей к файлам табличных пространств используется оператор ALTER TABLESPACE:

Листинг 10.2. Синтаксис оператора изменения табличного пространства ALTER TABLESPACE

```
ALTER TABLESPACE <имя табличного пространства> SET FILE [TO] '<путь к файлу>';
```

Эта операция не перемещает данные и не копирует файлы. Она применяется для изменения путей к табличным пространствам в случае их физического перемещения.

Для изменения путей к файлам табличных пространств необходимо указать путь к файлу. Можно указывать как абсолютный путь, так и относительный, в том числе с использованием псевдонима директории, заданного в directories.conf в секции tablespaces. В directories.conf указывается реальный путь к директории с псевдонимом <псевдоним директории>. Путь к файлу табличного пространства с использованием псевдонима директории указывается в формате: <псевдоним директории>/<файл>, например, dir/file.dat. Первый компонент пути (dir) будет обработан как псевдоним директории. Если псевдоним не будет найден в directories.conf, то путь будет обработан как относительный (относительно директории, в которой размещен основной файл базы данных). Все директории, используемые в пути, должны быть созданы заранее. В системную таблицу RDB$TABLESPACES в любом случае будет записан абсолютный путь к файлу табличного пространства.

Права на изменение табличных пространств есть только у администраторов и пользователей с привилегией ALTER ANY TABLESPACE.

Для удаления существующего табличного пространства используется оператор DROP TABLESPACE.

Листинг 10.3. Синтаксис оператора удаления табличного пространства DROP TABLESPACE

```
DROP TABLESPACE <имя табличного пространства>;
```

Если в удаляемом табличном пространстве хранятся таблицы или индексы, то будет выдано сообщение об ошибке.

Невозможно удаление табличного пространства, в котором размещено ограничение, при размещении соответствующей ему таблицы в другом табличном пространстве.

Права на удаление табличных пространств есть только у администраторов и пользователей с привилегией DROP ANY TABLESPACE.

10.1 Примечание для табличного пространства

Для существующего табличного пространства можно добавить комментарий, используя оператор COMMENT ON:

Листинг 10.4.

```
COMMENT ON
   TABLESPACE <имя табличного пространства> IS {'<текст примечания>' | NULL};
```

Значение NULL удаляет существующее примечание.

Выполнить оператор COMMENT ON TABLESPACE могут администраторы, владельцы домена или пользователи с привилегией ALTER ANY TABLESPACE.

Глава 11

Операторы DML

Для заполнения базы данных пользовательскими данными, изменения и удаления существующих данных используются операторы SQL подраздела DML (Data Manipulation Language). Операторы задают, что должно быть сделано с данными базы данных, не указывая, как именно это должно быть сделано.

Оператор `SELECT` — один из самых сложных и самых мощных операторов SQL в СУБД РЕД База Данных. Он позволяет выбирать данные из одной или нескольких таблиц на основании условий в предложении `WHERE`, условий объединения (оператор `SELECT` в предложении `UNION`) и условий соединения (предложение `ON`), если используется объединение (`UNION`) или соединение нескольких таблиц (`JOIN`).

Оператор выбирает данные из одной или нескольких таблиц, представлений или из хранимой процедуры выбора. Результатом выборки является выходной набор данных — множество строк одинаковой структуры, состав которых задан в списке выбора оператора `SELECT`.

Для добавления новых строк в таблицы или в представления базы данных используется оператор `INSERT`.

Для изменения данных в таблицах базы данных применяется оператор `UPDATE`.

Для изменения существующих строк в таблицах базы данных или для добавления новых данных, если такие строки еще не существуют, применяется оператор `UPDATE OR INSERT`.

Для удаления строк таблиц используется оператор `DELETE`.

Есть также оператор `EXECUTE BLOCK`, который позволяет в декларативной части SQL использовать некоторые императивные средства, применяемые в языке хранимых процедур и триггеров (PSQL).

Все действия по изменению данных выполняются в контексте (под управлением) какой-либо транзакции. Это может быть предварительно запущенная в клиенте оператором `SET TRANSACTION` транзакция с необходимыми характеристиками или транзакция по умолчанию, запускаемая системой автоматически при выполнении любых операций с данными и метаданными базы данных. О транзакциях см. в главе 5.

11.1 SELECT

Синтаксис оператора достаточно сложный. Он представлен в листинге 11.1. Далее в подразделе дается краткое описание структуры оператора `SELECT`, о полном описании всех предложении рассказывается в следующих подразделах этой главы.

Оператор (команда) `SELECT` извлекает данные из базы данных и передаёт их в приложение или в вызывающую SQL-команду. Данные возвращаются в виде набора строк, каждая строка содержит один или несколько столбцов и полей. Совокупность возвращаемых строк является результирующим набором данных команды.

Листинг 11.1. Синтаксис оператора выборки данных `SELECT`

```
[WITH [RECURSIVE] <CTE> [, <CTE> ...]]
SELECT
   [FIRST <значение>] [SKIP <значение>]
   [DISTINCT | ALL]
   <выходное поле> [, <выходное поле>]
FROM {<источники> | <соединения (joins)>}
[WHERE <условие выборки>]
[GROUP BY <условие группирования> [, <условие группирования> ...]
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
    [HAVING <условие выборки>]]
[WINDOW <спецификация окна> [, <спецификация окна> ...]]
[PLAN <выражение для плана поиска>]
[UNION [DISTINCT | ALL] <другой набор данных>]
[ORDER BY <выражение сортировки> [, <выражение сортировки> ...]]
[{ ROWS <m> [TO <n>]
   | [OFFSET <n> {ROW | ROWS}]
     [FETCH {FIRST | NEXT}
      [<m>] {ROW | ROWS} ONLY]
}]
[FOR UPDATE [OF <имя столбца> [, <имя столбца>]...]]
[WITH LOCK [SKIP LOCKED]]
[OPTIMIZE FOR {FIRST | ALL} ROWS]
[INTO [:]<переменная> [,[:]<переменная> ... ]]
```

Приведенный выше синтаксис не является полным синтаксисом SELECT. Он упрощен для документации и будет раскрыт в следующих разделах. Полный синтаксис SELECT можно найти ниже, в разделе Полный синтаксис SELECT.

Обязательными в команде SELECT являются список полей, запрашиваемых из базы данных, и ключевое слово FROM, за которым следует объект выборки (например, таблица).

В простейшей форме SELECT извлекает ряд полей из единственной таблицы, например:

```
SELECT id, name, address FROM contacts;
```

Или, для того чтобы извлечь все поля таблицы:

```
SELECT * FROM contacts;
```

Предложение WITH позволяет задать общее табличное выражение (CTE, Common Table Expression). Оно может быть рекурсивным (ключевое слово RECURSIVE) и обычным, не рекурсивным (значение по умолчанию).

Сразу после ключевого слова SELECT могут следовать предложения FIRST и SKIP, позволяющие определить количество строк, помещаемых в результирующий набор данных, выбранных на основании условий выборки (предложения ON, UNION и WHERE).

Ключевое слово DISTINCT указывает, что в выходной набор данных не должны входить дубликаты строк.

Далее идет сам список выбора, указывающий, какие столбцы из каких таблиц, участвующих в операции выборки, помещаются в выходной набор данных. Здесь могут располагаться константы, контекстные переменные и операторы SELECT, выбирающие из произвольных таблиц одно значение одного столбца.

Предложение FROM содержит список таблиц, из которых осуществляется выбор данных. В этом предложении может содержаться описание соединения (JOIN) нескольких таблиц для получения выходного набора данных.

Необязательное предложение WHERE задает условия выборки данных — те условия, которым должны удовлетворять строки исходной таблицы (исходных таблиц), для того, чтобы они попали в результирующий набор данных. Подробное описание условия выборки см. в разделе 11.1.7 этой главы.

Предложения GROUP BY и HAVING позволяют сгруппировать выбранные данные, если в списке выбора присутствуют агрегатные функции, обобщающие данные из нескольких строк исходной таблицы.

Предложение WINDOW предназначено для задания именованных окон, которые используются окон-

ными функциями.

Предложение UNION дает возможность объединить в выходном наборе данных несколько таблиц с одинаковой структурой.

В предложении PLAN можно задать свой план для выполнения запроса, который позволит ускорить процесс выбора данных.

Предложение ORDER BY задает упорядоченность выходного набора данных. Здесь также можно указать количество строк, которое должно быть помещено в результирующий набор данных (предложения ROWS, OFFSET, FETCH).

Предложение OPTIMIZE FOR позволяет задать необходимую стратегию оптимизации запросов для ускорения процесса выборки данных.

Необязательное предложение WITH LOCK запрещает параллельным процессам, транзакциям выполнять какие-либо изменения в данной таблице запроса. Подробности см. в главе 5.

Предложение SKIP LOCKED позволяет пропускать записи, заблокированные другими транзакциями, вместо того чтобы ждать их разблокирования или выдавать ошибку конфликта обновления.

С помощью предложения INTO в PSQL (хранимых процедурах, триггерах и др.) результаты выборки команды SELECT могут быть построчно загружены в локальные переменные (число, порядок и типы локальных переменных должны соответствовать полям SELECT).

11.1.1 WITH RECURSIVE

Предложение WITH позволяет задать общее табличное выражение (CTE, Common Table Expression). CTE описаны как виртуальные таблицы или представления, определённые в преамбуле основного запроса, которые участвуют в основном запросе. Основной запрос может ссылаться на любое CTE из определённых в преамбуле, как и при выборке данных из обычных таблиц или представлений. Оно может быть рекурсивным (ключевое слово RECURSIVE), то есть ссылающимся само на себя и обычным, не рекурсивным (значение по умолчанию). Синтаксис предложения представлен в листинге 11.2:

Листинг 11.2. Синтаксис предложения WITH RECURSIVE

```
WITH [RECURSIVE] <CTE> [, <CTE> ...]
SELECT ...
FROM ...

<CTE> ::= <псевдоним CTE> [(<список столбцов CTE>)] AS (<оператор SELECT или UNION>)

<список столбцов CTE> ::= <псевдоним столбца CTE> [, <псевдоним столбца CTE> ...]
```

Примечания по использованию CTE:

- Операторы WITH не могут быть вложенными;
- CTE могут использовать друг друга, но ссылки не должны иметь циклов;
- CTE могут быть использованы в любой части главного запроса или другого табличного выражения и сколько угодно раз;
- Основной запрос может ссылаться на CTE несколько раз, но с разными алиасами;
- CTE могут быть использованы в операторах INSERT, UPDATE и DELETE как подзапросы;
- Если CTE объявлен, то он должен быть обязательно использован;
- CTE могут быть использованы и в PSQL (FOR WITH ... SELECT ... INTO ...)

Пример не рекурсивного использования WITH:

```
WITH DEPT_YEAR_BUDGET AS (
   SELECT
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
      FISCAL_YEAR,
      DEPT_NO,
      SUM(PROJECTED_BUDGET) AS BUDGET
   FROM PROJ_DEPT_BUDGET
   GROUP BY FISCAL_YEAR, DEPT_NO )
SELECT
   D.DEPT_NO,
   D.DEPARTMENT,
   B_1993.BUDGET AS B_1993, B_1994.BUDGET AS B_1994,
   B_1995.BUDGET AS B_1995, B_1996.BUDGET AS B_1996
FROM DEPARTMENT D
   LEFT JOIN DEPT_YEAR_BUDGET B_1993
      ON D.DEPT_NO = B_1993.DEPT_NO
      AND B_1993.FISCAL_YEAR = 1993
   LEFT JOIN DEPT_YEAR_BUDGET B_1994
      ON D.DEPT_NO = B_1994.DEPT_NO
      AND B_1994.FISCAL_YEAR = 1994
   LEFT JOIN DEPT_YEAR_BUDGET B_1995
      ON D.DEPT_NO = B_1995.DEPT_NO
      AND B_1995.FISCAL_YEAR = 1995
   LEFT JOIN DEPT_YEAR_BUDGET B_1996
      ON D.DEPT_NO = B_1996.DEPT_NO
      AND B_1996.FISCAL_YEAR = 1996
WHERE EXISTS (SELECT * FROM PROJ_DEPT_BUDGET B
              WHERE D.DEPT_NO = B.DEPT_NO);
```

Рекурсивное (ссылающееся само на себя) CTE это объединение, у которого должен быть, по крайней мере, один не рекурсивный элемент, к которому привязываются остальные элементы объединения. Не рекурсивный элемент помещается в CTE первым. Рекурсивные члены отделяются от не рекурсивных и друг от друга с помощью UNION ALL. Объединение не рекурсивных элементов может быть любого типа.

Рекурсивное CTE требует наличия ключевого слова RECURSIVE справа от WITH. Каждый рекурсивный член объединения может сослаться на себя только один раз и это должно быть сделано в предложении FROM.

Главным преимуществом рекурсивных CTE является то, что они используют гораздо меньше памяти и процессорного времени, чем эквивалентные рекурсивные хранимые процедуры.

Пример рекурсивного использования WITH:

```
WITH RECURSIVE DEPT_YEAR_BUDGET AS (
   SELECT
      FISCAL_YEAR,
      DEPT_NO,
      SUM(PROJECTED_BUDGET) AS BUDGET
   FROM PROJ_DEPT_BUDGET
   GROUP BY FISCAL_YEAR, DEPT_NO ),
DEPT_TREE AS (
   SELECT
      DEPT_NO,
      HEAD_DEPT,
      DEPARTMENT,
      CAST('' AS VARCHAR(255)) AS INDENT
   FROM DEPARTMENT
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
    WHERE HEAD_DEPT IS NULL
    UNION ALL
    SELECT
       D.DEPT_NO,
       D.HEAD_DEPT,
       D.DEPARTMENT,
       H.INDENT || ' '
    FROM DEPARTMENT D
    JOIN DEPT_TREE H ON D.HEAD_DEPT = H.DEPT_NO )
SELECT
   D.DEPT_NO,
   D.INDENT || D.DEPARTMENT AS DEPARTMENT,
   B_1993.BUDGET AS B_1993, B_1994.BUDGET AS B_1994,
   B_1995.BUDGET AS B_1995, B_1996.BUDGET AS B_1996
FROM DEPT_TREE D
   LEFT JOIN DEPT_YEAR_BUDGET B_1993
      ON D.DEPT_NO = B_1993.DEPT_NO
      AND B_1993.FISCAL_YEAR = 1993
   LEFT JOIN DEPT_YEAR_BUDGET B_1994
      ON D.DEPT_NO = B_1994.DEPT_NO
      AND B_1994.FISCAL_YEAR = 1994
   LEFT JOIN DEPT_YEAR_BUDGET B_1995
      ON D.DEPT_NO = B_1995.DEPT_NO
      AND B_1995.FISCAL_YEAR = 1995
   LEFT JOIN DEPT_YEAR_BUDGET B_1996
      ON D.DEPT_NO = B_1996.DEPT_NO
      AND B_1996.FISCAL_YEAR = 1996;
```

Примечания для рекурсивного CTE:

- В рекурсивных членах объединения не разрешается использовать агрегаты (DISTINCT, GROUP BY, HAVING) и агрегатные функции (SUM, COUNT, MAX и т.п.);
- Рекурсивная ссылка не может быть участником внешнего объединения OUTER JOIN;
- Максимальная глубина рекурсии составляет 1024;
- Рекурсивный член не может быть представлен в виде производной таблицы.

11.1.2 Список выбора

После ключевого слова SELECT может следовать какие столбцы исходных таблиц должны присутствовать в выходном наборе данных.

Листинг 11.3. Синтаксис списка выбора в операторе SELECT

```
SELECT ...
   [{ ALL | DISTINCT }] <список выбора>
FROM ...

<список выбора> ::= * | <подсписок> [, <подсписок> ...]

<подсписок> ::=
  <спецификатор>.*
| <выражение> [[AS] <столбец>]
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
<выражение> ::=
  [спецификатор.]<столбец таблицы (представления) или псевдоним>
| [спецификатор.]<выходной параметр селективной хранимой процедуры>
| <литерал>
| <контекстная переменная>
| <вызов функции>
| <подзапрос SELECT, возвращающий единственное скалярное значение>
| <конструкция CASE>
| NEXT VALUE FOR <имя генератора>
| NULL
| <любое выражение, возвращающее единственное значение>

<вызов функции> ::=
  <скалярная функция>
| <агрегатная функция>
| <оконная функция>

<спецификатор> ::= имя таблицы (представления) или псевдоним таблицы (представления,
хранимой процедуры, производной таблицы)
```

Символ "*" означает, что в результирующем наборе данных должны присутствовать все столбцы исходной таблицы (исходных таблиц).

Если в запросе используется несколько таблиц, у которых имена столбцов могут совпадать, то слева от имени нужного столбца прибавляется <спецификатор> — имя таблицы или псевдоним таблицы и точка. Такая конструкция называется уточненным именем столбца:

```
select COUNTRY.CODCOUNTRY
from country;
```

Для любого столбца в списке выбора можно задать псевдоним после ключевого слова AS. Псевдонимом может быть любой правильный идентификатор. Допустимо также использование имен с ограничителями. Например,

```
select COUNTRY.CODCOUNTRY AS "Код страны"
from country;
```

Если для отдельных столбцов были заданы псевдонимы, то при использовании для отображения строк таблицы утилиты isql (или менеджера баз данных с графическим интерфейсом) именно псевдонимы столбцов, а не их имена в таблицах базы данных будут присутствовать в заголовках столбцов. Кроме того, псевдонимы столбцов могут быть использованы в предложениях ORDER BY и GROUP BY. Если для столбцов были заданы псевдонимы, то в большинстве конструкций оператора SELECT могут быть использованы как имена столбцов, так и их псевдонимы.

Алиасы (псевдонимы) заменяют оригинальное имя таблицы/представления/хранимой процедуры: как только определён алиас для соответствующего отношения, использовать оригинальное имя нельзя.

Перед списком столбцов можно указать необязательное ключевое слово DISTINCT или ALL:

- DISTINCT исключает дубликаты строк. То есть, если несколько строк имеют одинаковые значения в каждом столбце, только одна из них будет включена в результирующий набор;
- ALL используется по умолчанию: возвращаются все строки, включая дубликаты.

Выражение COLLATE не изменяет содержимое поля, однако, если указать COLLATE для опреде-

лённого поля, то это может изменить чувствительность к регистру символов или к акцентам (accent sensitivity), что, в свою очередь, может повлиять на:

- Порядок сортировки, если это поле указано в выражении ORDER BY;
- Группировку, если это поле указано в выражении GROUP BY;
- Количество возвращаемых строк, если используется DISTINCT.

11.1.3 DISTINCT, ALL

Перед списком полей могут быть добавлены ключевые слова DISTINCT или ALL.

Листинг 11.4. Синтаксис предложения DISTINCT, ALL

```
SELECT
   [DISTINCT | ALL] <выходное поле> [, <выходное поле> ...]
...
FROM ...
```

Ключевое слово DISTINCT исключает дубликаты строк из результирующего набора. То есть, если несколько строк имеют одинаковые значения в каждом столбце, только одна из них будет включена в результирующий набор.

Ключевое слово ALL, используемое по умолчанию, означает, что в результирующий набор данных попадут все строки исходных таблиц, которые соответствуют условиям выборки (предложение ON в соединении таблиц и предложение WHERE), в том числе и дубликаты.

11.1.4 FIRST и SKIP

После ключевого слова SELECT может следовать указание, какое количество полученных при выполнении этого оператора строк исходной таблицы (исходных таблиц) должно помещаться в выходной набор данных.

Листинг 11.5. Синтаксис предложения FIRST, SKIP

```
SELECT
   [FIRST <значение>] [SKIP <значение>]
   FROM ...
   ...

<значение> ::=
     <целочисленный литерал>
   | <параметр запроса>
   | (<выражение>)
```

В качестве <значение> может выступать целочисленный литерал, параметр запроса (? — в DSQL и :paramname — в PSQL) или выражение, возвращающее целочисленное значение. Если возвращается дробное число, то десятичные знаки просто отбрасываются без округления. Любой аргумент FIRST и SKIP, не являющийся целочисленным литералом или параметром SQL, должен быть заключен в круглые скобки. Это означает, что выражение подзапроса должно быть заключено в две пары круглых скобок. Отрицательные значения в SKIP и FIRST приводят к ошибке.

Предложения FIRST и SKIP не входят в стандарт SQL. Используйте стандартные предложения OFFSET и FETCH везде, где это возможно.

Предложение FIRST m ограничивает вывод запроса первыми m строками. Нумерация строк начинается с 1. Если вывод запроса содержит меньше m строк, то в результирующий набор войдут все вернувшиеся строки, без каких-либо сообщений. FIRST 0 допускается и возвращает пустой набор.

Предложение SKIP n пропускает первые n строк в выводе запроса, исключая их из результирующего набора. Если вывод запроса содержит меньше n строк, то результирующий набор будет пустым. SKIP 0 допускается, но не имеет смысла.

В одном операторе SELECT можно указать сразу и FIRST, и SKIP. При совместном использовании, например, FIRST m SKIP n, n первых строк вывода отбрасываются и возвращаются первые m строк оставшейся части.

Например, если задать

```
SELECT FIRST 10 ...
```

то в набор данных будут помещены первые 10 выбранных строк.

Если в операторе указать

```
SELECT SKIP 9 ...
```

то в набор данных будут помещены строки, начиная с десятой.

Если же задать

```
SELECT FIRST 10 SKIP 9 ...
```

то в набор данных будет помещены 10 строк, начиная с 10-ой.

Например, если нужно выбрать только первую половину строк из таблицы FIRM, то можно задать предложение FIRST в виде следующего выражения:

```
SELECT FIRST ((SELECT COUNT (*) FROM COUNTRY) / 2) ...
```

Использование ключевых слов FIRST и/или SKIP не требует обязательного присутствия в операторе SELECT предложения ORDER BY, как в случае использования предложения ROWS.

Ключевые слова FIRST и SKIP не могут присутствовать в операторе SELECT, где указано предложение ROWS.

Количество помещаемых в результирующий набор данных строк может также задаваться (корректироваться) при использовании предложений ORDER BY и ROWS, а также предложений OFFSET, FETCH.

11.1.5 FROM

Предложение FROM задает источники, из которых осуществляется выборка данных. В качестве источников могут выступать таблицы, представления, хранимые процедуры выбора, получающие данные из одной или нескольких таблиц, или общее табличное выражение (CTE). Эти таблицы, представления, процедуры могут комбинироваться с использованием разнообразных видов соединений (JOIN), но об этом будет рассказано в другом разделе.

Листинг 11.6. Синтаксис предложения FROM в операторе SELECT

```
SELECT
...
   FROM <источник> [, <источник> ...]
[...]
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
<источник> ::= <табличный источник> | <соединение>

<табличный источник> ::=
  <источник или CTE> [[AS] псевдоним]
| [LATERAL] <производная таблица> [ [AS] <псевдоним> [(<список столбцов>)]]
| <родительская соединённая таблица>

<источник или CTE> ::=
  <таблица или представление>
| CTE
| [имя пакета.]<имя процедуры> [(<аргументы процедуры>)]

<аргументы процедуры> ::= <значение> [, <значение> ...]

<производная таблица> ::=
   (<SELECT запрос>) [[AS] <псевдоним производной таблицы>]
   [(<псевдоним столбца> [, <псевдоним столбца>])]

<список столбцов> ::= <столбец> [, <столбец> ...]

<столбец> ::=
  <таблица или псевдоним>
| <представление>
| <CTE>
| <производная таблица>
```

Предложение FROM позволяет указать несколько таблиц (представлений, хранимых процедур выбора), разделенных запятыми, где каждая задает соединяемую таблицу. Это соответствует неявному внутреннему соединению (INNER JOIN). Здесь условие соединения таблиц задается не предложением ON, как при явном соединении, а присутствует в предложении WHERE. Не рекомендуется использовать неявное соединение таблиц. Следует явно задавать соединяемые таблицы (в том числе, полученные из представлений или из хранимых процедур выбора) с использованием ключевого слова JOIN и условия соединения, указанного в предложении ON. Подробнее о соединении таблиц см. в разделе 11.1.6.

Если в предложении FROM для таблицы (представления или хранимой процедуры) указан псевдоним, то во всех конструкциях оператора SELECT должен обязательно использоваться именно этот псевдоним. Использование имени таблицы в этом случае недопустимо.

Например, следующий оператор не будет выполнен (вы получите сообщение об ошибке):

```
SELECT COUNTRY.CODCOUNTRY
FROM COUNTRY C;
```

Выборка из таблицы или представления

При выборке из таблицы или представления предложение FROM не требует ничего, кроме имени. Псевдоним (алиас) может быть полезен или даже необходим, если есть подзапросы, которые ссылаются на основной оператор select. Такие подзапросы называются коррелированными подзапросами.

Если для таблицы или представления задан псевдоним, необходимо всегда использовать этот псевдоним вместо имени таблицы при запросах (и везде, где есть ссылки на столбцы, например, в

предложениях ORDER BY, GROUP BY и WHERE).

```
SELECT id, name, sex, age
FROM actors
WHERE state = 'Ohio';
```

Выборка из селективной хранимой процедуры

Селективная хранимая процедура (т.е. с возможностью выборки) должна содержать хотя бы один выходной параметр и использовать ключевое слово SUSPEND, чтобы вызывающая сторона могла получать выходные строки по одной, как при выборе из таблицы или представления.

Выходные параметры селективной хранимой процедуры с точки зрения команды SELECT соответствуют полям обычной таблицы.

Выборка из хранимой процедуры без входных параметров осуществляется точно так же, как обычная выборка из таблицы. Если хранимая процедура требует входные параметры, то они должны быть указаны в скобках после имени процедуры:

```
SELECT name, az, alt
FROM visible_stars('Brugge' , current_date, '22:30');
```

Значения для необязательных параметров (то есть, параметров, для которых определены значения по умолчанию) могут быть указаны или опущены. Однако если параметры задаются частично, то пропущенные параметры должны быть в конце перечисления внутри скобок.

Выборка из производной таблицы

В настоящей версии РЕД Базы Данных в предложении FROM можно создавать производные таблицы (derived tables), которые могут использоваться и в других предложениях того же оператора SELECT.

Синтаксис создания производной таблицы в предложении FROM показан в листинге 11.7.

Листинг 11.7. Синтаксис производной таблицы

```
<производная таблица> ::= (<оператор SELECT>) [[AS] <псевдоним таблицы>]
                          [(<псевдоним столбца> [, <псевдоним столбца>...])]
```

Возвращаемый набор данных такого оператора представляет собой виртуальную таблицу, к которой можно составлять запросы так, как будто это обычная таблица.

Сам оператор SELECT, который создает производную таблицу, заключается в круглые скобки. Это может быть оператор любой сложности. После него идет ключевое слово AS, за которым следует имя псевдонима производной таблицы. По этому имени можно обращаться к производной таблице из любого оператора, как если бы это была таблица, хранящаяся в базе данных, или представление, использующее одну или несколько таблиц базы данных, или хранимая процедура выбора.

После псевдонима таблицы в круглых скобках можно указать список псевдонимов столбцов, которые были указаны в списке выбора этого оператора SELECT, создающего (описывающего) производную таблицу. Количество столбцов в списке выбора оператора SELECT и количество псевдонимов столбцов должно быть одинаковым. Дальнейшее обращение к именам столбцов должно выполняться только с использованием имен псевдонимов, если они были указаны.

Ключевое слово LATERAL обозначает таблицу как боковую производную таблицу. Боковые производные таблицы могут ссылаться на таблицы (включая другие производные таблицы), которые встречаются ранее в предложении FROM.

Пример.

Следующий оператор выбирает список имен и внутренних идентификаторов всех несистемных таблиц из системной таблицы RDB$RELATIONS. Пример использования производной таблицы:

```
SELECT *
FROM (SELECT RDB$RELATION_NAME,
             RDB$RELATION_ID
      FROM RDB$RELATIONS
      WHERE RDB$RELATION_NAME NOT STARTING WITH 'RDB$')
AS R ("Таблица" , "Идентификатор");
```

Оператор отбирает только те таблицы, имена которых не начинаются с символов 'RDB$', то есть таблицы, созданные пользователем, а не системой. Полученному набору данных присваивается имя R. Это имя может в дальнейшем использоваться в данном операторе как имя таблицы базы данных. Двум выбираемым в операторе столбцам присваиваются псевдонимы "Таблица" и "Идентификатор". Только по заданным псевдонимам можно будет в этом операторе обращаться к этим столбцам.

Примечания для производных таблиц:

- Производные таблицы могут быть вложенными.
- Производные таблицы могут быть объединениями и использоваться в объединениях. Они могут содержать агрегатные функции, подзапросы и соединения, и сами по себе могут быть использованы в агрегатных функциях, подзапросах и соединениях. Они также могут быть хранимыми процедурами или запросами из них. Они могут иметь предложения WHERE, ORDER BY и GROUP BY, указания FIRST, SKIP или ROWS и т.д.
- Каждый столбец в производной таблице должен иметь имя. Если этого нет по своей природе (например, потому что это — константа), то надо в обычном порядке присвоить псевдоним или добавить список псевдонимов столбцов в спецификации производной таблицы.
- Список псевдонимов столбцов опциональный, но если он присутствует, то должен быть полным (т.е. он должен содержать псевдоним для каждого столбца производной таблицы).
- Оптимизатор может обрабатывать производные таблицы очень эффективно. Однако если производная таблица включена во внутреннее соединение и содержит подзапрос, то никакой порядок соединения не может быть использован оптимизатором.

Выборка из общих табличных выражений

Общие табличные выражения являются более сложной и более мощной вариацией производных таблиц. CTE состоят из преамбулы, начинающейся с ключевого слова WITH, которая определяет одно или более общих табличных выражений (каждое из которых может иметь список алиасов полей). Основной запрос, который следует за преамбулой, может обращаться к CTE так, как будто это обычные таблицы. CTE доступны только в рамках основного запроса.

Подробно CTE описываются в разделе 11.1.1.

Пример.

Предположим, что у нас есть таблица COEFFS, которая содержит коэффициенты для ряда квадратных уравнений, которые мы собираемся решить. В зависимости от значений коэффициентов a, b и c, каждое уравнение может иметь ноль, одно или два решения. Можно найти эти решения с помощью CTE.

```
WITH vars (b, D, denom) AS
          (SELECT b, b*b - 4*a*c, 2*a
           FROM coeffs ),
     vars2 (b, D, denom, sqrtD) AS
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
            (SELECT b, D, denom, IIF (D >= 0, sqrt(D), NULL)
             FROM vars )
SELECT
     IIF (D >= 0, (-b - sqrtD) / denom, NULL) AS sol_1,
     IIF (D > 0, (-b + sqrtD) / denom, NULL) AS sol_2
FROM vars2;
```

11.1.6 JOIN

Соединения объединяют данные из двух источников в один набор. Объединение выполняется по каждой строке и обычно включает проверку условия объединения, чтобы определить, какие строки должны быть объединены и включены в результирующий набор данных. Существует несколько типов (`INNER`, `OUTER`) и классов (квалифицированные, естественные и т. д.) соединений, каждый из которых имеет свой собственный синтаксис и правила.

Поскольку соединения можно объединять в цепочки, наборы данных, участвующие в соединении, сами могут быть соединенными наборами.

Синтаксис предложения `JOIN`:

Листинг 11.8. Синтаксис предложения `JOIN`

```
SELECT
...
   FROM <источник> [, <источник> ...]
[...]

<источник> ::= <табличный источник> | <соединение>

<табличный источник> ::=
  <источник или CTE> [[AS] псевдоним]
| [LATERAL] <производная таблица> [ [AS] <псевдоним> [(<список столбцов>)]]
| <родительская соединённая таблица>

<источник или CTE> ::=
  <таблица или представление>
| CTE
| [имя пакета.]<имя процедуры> [(<аргументы процедуры>)]

<аргументы процедуры> ::= <значение> [, <значение> ...]

<производная таблица> ::=
   (<SELECT запрос>) [[AS] <псевдоним производной таблицы>]
   [(<псевдоним столбца> [, <псевдоним столбца>])]

<список столбцов> ::= <столбец> [, <столбец> ...]

<столбец> ::=
  <таблица или псевдоним>
| <представление>
| <CTE>
| <производная таблица>
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
<родительская соединённая таблица> ::=
  (<родительская соединённая таблица>)
| (<соединение>)

<соединение> ::=
  <перекрёстное соединение>
| <естественное соединение>
| <квалифицированное соединение>

<перекрёстное соединение> ::=
   <источник> CROSS JOIN <табличный источник>

<естественное соединение> ::=
   <источник> NATURAL [<тип соединения>] JOIN <табличный источник>

<тип соединения> ::= INNER | { LEFT | RIGHT | FULL } [OUTER]

<квалифицированное соединение> ::=
   <источник> [<тип соединения>] JOIN <табличный источник>
   { ON <условие соединения>
   | USING (<список столбцов>) }
```

Если задается предложение `USING`, то предложение `ON` должно отсутствовать. В предложении `USING` перечисляются имена столбцов, которые должны присутствовать в обеих соединяемых таблицах. В этом случае условием соединения является равенство значений указанных столбцов в таблицах.

Неявное соединение

Неявное соединение дает декартово произведение двух множеств строк обеих соединяемых таблиц. Каждая строка левой таблицы соединяется с каждой строкой правой таблицы. Но если в предложении `WHERE` добавить условие соединения, то результат эквивалентен операции `INNER JOIN` с таким же условием.

Листинг 11.9. Синтаксис неявного соединения

```
SELECT ...
FROM <источник> , <источник> [, <источник>...]
[WHERE <условие>]
```

В настоящее время синтаксис неявных соединений не рекомендуется к использованию.

Внутреннее (INNER) соединение

Для внутреннего (`INNER JOIN`) соединения тип соединения можно не указывать — соединение является внутренним по умолчанию.

Результат соединения логически формируется следующим образом: каждая строка одной таблицы сопоставляется с каждой строкой второй таблицы, после чего для полученной "соединённой" строки проверяется условие соединения (заданное в предложении `ON`). Если условие истинно, в таблицу-результат добавляется соответствующая "соединённая" строка.

Порядок таблиц для оператора неважен, поскольку оператор является коммутативным.

Пример.

Предположим, у нас есть таблица PEOPLE:

COD	NAME	SEX
34	Ivan	0
56	Ruslan	0
109	Ann	1

и таблица STAFF:

CODPEOPLE	CODORG	DUTIES
56	12346	accountant
78	35456	programmer
109	46743	manager

Следующий запрос соединит таблицы:

```
SELECT *
FROM PEOPLE
    JOIN STAFF ON PEOPLE.COD = STAFF.CODPEOPLE;

COD NAME   SEX CODPEOPLE CODORG DUTIES
=== ====== === ========= ====== ==========
56  Ruslan 0   56        12346  accountant
109 Ann    1   109       46743  manager
```

Можно указать слово INNER, но обычно оно опускается.

Точно такой же результат можно получить, выполнив следующий оператор SELECT:

```
SELECT *
FROM PEOPLE, STAFF
WHERE PEOPLE.COD = STAFF.CODPEOPLE;
```

Это неявное внутреннее соединение. Условие соединения в этом случае задается в предложении WHERE.

Внешние (OUTER) соединения

В результат внешнего соединения обязательно входят все строки либо одной, либо обеих таблиц. Внешние соединения (OUTER JOIN) бывают левыми (LEFT), правыми (RIGHT) и полными (FULL). При указании слов LEFT, RIGHT, FULL слово OUTER можно опустить.

Левое внешнее соединение

Результат левого внешнего соединения (LEFT OUTER JOIN) логически формируется следующим образом: в результат включается внутреннее соединение (INNER JOIN) левой и правой таблиц по условию (заданному в предложении ON). Затем в результат добавляются те строки левой таблицы, которые не вошли во внутреннее соединение. Для таких строк столбцы, соответствующие правой таблице, заполняются значениями NULL.

Порядок таблиц для оператора важен, поскольку оператор не является коммутативным.

Пример.

Пусть требуется для каждого сотрудника таблицы STAFF по коду получить его имя из таблицы PEOPLE. Следующий оператор выполняет необходимую выборку:

```
SELECT
   PEOPLE.NAME AS "Сотрудник",
   SEX AS "Пол",
   DUTIES AS "Должность"
FROM STAFF
LEFT OUTER JOIN PEOPLE
   ON STAFF.CODPEOPLE = PEOPLE.COD;

Сотрудник Пол    Должность
========= ====== ==========
Ruslan    0      accountant
<null>    <null> programmer
Ann       1      manager
```

Слово OUTER можно опустить.

Правое внешнее соединение

Правое внешнее соединение (RIGHT OUTER JOIN) отличается от левого внешнего соединения только порядком выполнения соединения таблиц. При левом внешнем соединении действия выполняются слева направо, при правом же внешнем соединении наоборот — справа налево.

Результат правого внешнего соединения логически формируется следующим образом: в результат включается внутреннее соединение (INNER JOIN) левой и правой таблиц по условию (заданному в предложении ON). Затем в результат добавляются те строки правой таблицы, которые не вошли во внутреннее соединение. Для таких строк столбцы, соответствующие левой таблице, заполняются значениями NULL.

Порядок таблиц для оператора важен, поскольку оператор не является коммутативным.

Пример.

Пусть требуется для каждого человека таблицы PEOPLE по коду получить его должность из таблицы STAFF. Следующий оператор выполняет необходимую выборку:

```
SELECT
   PEOPLE.NAME AS "Сотрудник",
   SEX AS "Пол",
   DUTIES AS "Должность"
FROM STAFF
RIGHT JOIN PEOPLE
   ON STAFF.CODPEOPLE = PEOPLE.COD;

Сотрудник Пол    Должность
========= ====== ==========
Ivan      0      <null>
Ruslan    0      accountant
Ann       1      manager
```

Полное внешнее соединение

Результат полного внешнего соединения (FULL OUTER JOIN) логически формируется следующим образом: в результат включается внутреннее соединение (INNER JOIN) левой и правой таблиц по условию (заданному в предложении ON). Затем добавляются те строки первой таблицы, которые не вошли во внутреннее соединение. Для таких строк столбцы, соответствующие второй таблице, заполняются значениями NULL. И потом добавляются те строки второй таблицы, которые не вошли во внутреннее соединение на шаге 1. Для таких строк столбцы, соответствующие первой таблице, заполняются значениями NULL.

Порядок таблиц для оператора неважен, поскольку оператор является коммутативным.

Пример.

Пусть требуется для каждого человека таблицы PEOPLE по коду получить его должность из таблицы STAFF и для каждого сотрудника таблицы STAFF по коду получить его имя из таблицы PEOPLE. Следующий оператор выполняет необходимую выборку:

```
SELECT
   PEOPLE.NAME AS "Сотрудник",
   SEX AS "Пол",
   DUTIES AS "Должность"
FROM STAFF
FULL OUTER JOIN PEOPLE
   ON STAFF.CODPEOPLE = PEOPLE.COD;

Сотрудник Пол    Должность
========= ====== ==========
Ivan      0      <null>
Ruslan    0      accountant
Ann       1      manager
<null>    <null> programmer
```

Перекрестное (CROSS) соединение

Перекрестное соединение (CROSS JOIN) дает декартово произведение двух множеств строк обеих соединяемых таблиц. Каждая строка левой таблицы соединяется с каждой строкой правой таблицы.

Листинг 11.10. Синтаксис перекрестного соединения

```
SELECT ...
FROM <источник> CROSS JOIN <источник>
```

Порядок таблиц для оператора неважен, поскольку оператор является коммутативным.

Следующие три конструкции эквивалентны.

```
FROM <таблица1> CROSS JOIN <таблица2>
FROM <таблица1>, <таблица2>
FROM <таблица1> INNER JOIN <таблица2> ON <всегда истинное условие>
```

Пример.

```
SELECT
   PEOPLE.NAME AS "Сотрудник",
   SEX AS "Пол",
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
   DUTIES AS "Должность"
FROM STAFF
   CROSS JOIN PEOPLE;

Сотрудник Пол    Должность
========= ====== ==========
Ivan      0      accountant
Ruslan    0      accountant
Ann       1      accountant
Ivan      0      programmer
Ruslan    0      programmer
Ann       1      programmer
Ivan      0      manager
Ruslan    0      manager
Ann       1      manager
```

Квалифицированные соединения

Квалифицированные соединения задают условия для объединения строк. Это происходит либо в явном виде в предложении `ON`, либо в неявном виде в предложении `USING`.

```
<квалифицированные соединения> ::=
   <источник> [<тип соединения>] JOIN <табличный источник>
   { ON <условие соединения>
   | USING (<список столбцов>) }

<тип соединения> ::= INNER | {LEFT | RIGHT | FULL} [OUTER]
```

Соединения именованными столбцами

В синтаксисе явного соединения есть предложение `ON` с условием соединения, в котором может быть указано любое логическое выражение, но, как правило, оно содержит условие сравнения между двумя участвующими источниками. Довольно часто, это условие — проверка на равенство. Такие соединения называются эквисоединениями.

Эквисоединения часто сравнивают столбцы, которые имеют одно и то же имя в обеих таблицах. Для таких соединений мы можем использовать второй тип явных соединений, называемый соединением именованными столбцами (`Named Columns Joins`). Соединение именованными столбцами осуществляются с помощью предложения `USING`, в котором перечисляются только имена столбцов.

Соединения именованными столбцами доступны только в диалекте 3.

Листинг 11.11. Синтаксис соединения именованными столбцами

```
SELECT ...
FROM <источник> [<тип соединения>] JOIN <источник> USING (<список столбцов>)

<тип соединения> ::= INNER | {LEFT | RIGHT | FULL} [OUTER]
```

Пример.

Следующий запрос:

```
SELECT *
FROM flotsam f
  JOIN jetsam j
     ON f.sea = j.sea AND f.ship = j.ship;
```

можно переписать, используя предложение USING:

```
SELECT *
FROM flotsam
  JOIN jetsam USING (sea, ship);
```

Результирующий набор в этом примере будет различаться. Используя предложение ON, выборка будет содержать каждый из столбцов SEA и SHIP дважды: один раз для таблицы FLOTSAM и один раз для таблицы JETSAM. Очевидно, что они будут иметь они и те же значения. Результат соединения именованными столбцами, с помощью предложения USING, будет содержать эти столбцы один раз.

Для внешних (OUTER) соединений именованными столбцами существуют дополнительные нюансы при использовании SELECT * или неполного имени столбца. Если столбец строки из одного источника не имеет совпадений со столбцом строки из другого источника, но все равно должен быть включён результат из-за инструкций LEFT, RIGHT или FULL, то объединяемый столбец получит не NULL значение. Это достаточно справедливо, но теперь вы не можете сказать из какого набора левого, правого или обоих пришло это значение. Это особенно обманывает, когда значения пришли из правой части набора данных, потому что "*" всегда отображает для комбинированных столбцов значения из левой части набора данных, даже если используется RIGHT соединение. Лучше избегать "*" в серьёзных запросах и перечислять все имена столбцов для соединяемых множеств.

Соединения с явным условием

В синтаксисе явного соединения есть предложение ON, с условием соединения, в котором может быть указано любое логическое выражение, но, как правило, оно содержит условие сравнения между двумя участвующими источниками.

Довольно часто, это условие - проверка на равенство (или ряд проверок на равенство объединённых оператором AND) использующая оператор "=". Такие соединения называются эквисоединениями.

Примеры соединений с явными условиями:

```
 /*
* Выборка всех заказчиков из города Детройт, которые
* сделали покупку.
*/
SELECT *
FROM customers c
 JOIN sales s ON s.cust_id = c.id
WHERE c.city = 'Detroit'

/*
* Тоже самое, но включает в выборку заказчиков, которые
* не совершали покупки.
*/
SELECT *
FROM customers c
LEFT JOIN sales s ON s.cust_id = c.id
WHERE c.city = 'Detroit'
```

Естественное соединение

Естественное соединение выполняет эквисоединение по всем одноименным столбцам правой и левой таблицы. Типы данных этих столбцов должны быть совместимыми. Естественные соединения доступны только в диалекте 3.

Листинг 11.12. Синтаксис естественного соединения

```
SELECT ...
FROM <источник> NATURAL [<тип соединения>] JOIN <источник>

<тип соединения> ::= INNER | {LEFT | RIGHT | FULL} [OUTER]
```

Пример.

Создадим две таблицы с некоторыми одноименными столбцами:

```
CREATE TABLE Table1 (
   a BIGINT,
   s VARCHAR(12),
   ins_date DATE );

CREATE TABLE Table2 (
   a BIGINT,
   b VARCHAR(12),
   x FLOAT,
   ins_date DATE );
```

Естественное соединение таблиц будет происходить по столбцам `a` и `ins_date` и два следующих оператора дадут один и тот же результат:

```
SELECT *
FROM Table1
   NATURAL JOIN Table2;

SELECT *
FROM Table1
   JOIN Table2 USING (a, ins_date);
```

Как и все соединения, естественные соединения являются внутренними соединениями по умолчанию, но можно превратить их во внешние соединения, указав `LEFT, RIGHT` или `FULL` перед ключевым словом `JOIN`.

> Если в двух исходных таблицах не будут найдены одноименные столбцы, то будет выполнен `CROSS JOIN`.

Смешивание явного и неявного соединения

Смешивание явных и неявных соединений не рекомендуется, но позволяется. Некоторые виды смешивания запрещены в РЕД Базе Данных.

Например, такой запрос вызовет ошибку `"Column unknown -TA.COL1"`:

```
SELECT *
FROM TA, TB
   JOIN TC ON TA.COL1 = TC.COL1
WHERE TA.COL2 = TB.COL2;
```

Это происходит потому, что явный JOIN не может видеть таблицу TA. Однако следующий запрос будет выполнен без ошибок, поскольку изоляция не нарушена.

```
SELECT *
FROM TA, TB
   JOIN TC ON TB.COL1 = TC.COL1
WHERE TA.COL2 = TB.COL2;
```

Примечание о равенстве

> Это примечание об операторах равенства и неравенства применимо везде в языке SQL РЕД Базы Данных, а не только в условиях JOIN.

Оператор «=», который явно используется во многих условных соединениях и неявно в соединениях по именованным столбцам и естественных соединениях, сопоставляет только значения со значениями. Согласно стандарту SQL, NULL не является значением, и, следовательно, два NULL не равны и не неравны друг другу. Если нужно, чтобы NULL совпадали друг с другом в объединении, используйте оператор IS NOT DISTINCT FROM. Этот оператор возвращает true, если операнды имеют одинаковое значение или если они оба NULL.

```
select *
   from A join B
   on A.id is not distinct from B.code;
```

Аналогично, если требуется объединить по признаку неравенства, используйте IS DISTINCT FROM, а не «<>», если нужно, чтобы NULL считался отличающимся от любого значения, а два NULL считались равными:

```
select *
   from A join B
   on A.id is distinct from B.code;
```

Неоднозначные имена полей в соединениях

РЕД База Данных отвергает неполные имена полей в запросе, если эти имена полей существуют в более чем одном наборе данных, участвующих в объединении. Это также верно для внутренних эквисоединений, в которых имена полей фигурируют в предложении ON:

```
SELECT a, b, c
   FROM TA
   JOIN TB ON TA.a = TB.a
```

Существует одно исключение из этого правила: соединения по именованным столбцам и естественные соединения, которые используют неполное имя поля в процессе подбора, могут использоваться законно. Это же относится и к одноименным объединяемым столбцам. Для соединений по именованным столбцам эти столбцы должны быть перечислены в предложении USING. Для естественных

соединений это столбцы, имена которых присутствуют в обеих таблицах. Особенно во внешних соединениях, плоское имя `colname` является не всегда тем же самым что `left.colname` или `right.colname`. Типы данных могут отличаться, и один из полных столбцов может иметь значение `NULL`, в то время как другой нет. В этом случае значение в объединённом, неполном столбце может замаскировать тот факт, что одно из исходных значений отсутствует.

Соединения с хранимыми процедурами

Если соединение происходит с хранимой процедурой, которая не коррелирована с другими потоками данных через входные параметры, то нет никаких особенностей. В противном случае есть одна особенность: потоки, используемые во входных параметрах, должны быть описаны раньше соединения с хранимой процедурой:

```
SELECT *
   FROM MY_TAB
   JOIN MY_PROC(MY_TAB.F) ON 1 = 1
```

Запрос, написанный следующим образом вызовет ошибку:

```
SELECT *
   FROM MY_PROC(MY_TAB.F)
   JOIN MY_TAB ON 1 = 1
```

Соединение с производными таблицами

Производная таблица, определенная с помощью ключевого слова `LATERAL`, называется латеральной производной таблицей. Если производная таблица определена как латеральная, то разрешается ссылаться на другие таблицы в том же предложении `FROM`, но только на те, которые были объявлены раньше в предложении `FROM`. Без `LATERAL` каждый подзапрос выполняется независимо и поэтому не может обращаться к другим элементам `FROM`.

Элемент `LATERAL` может находиться на верхнем уровне списка `FROM` или в дереве `JOIN`. В последнем случае он может также ссылаться на любые элементы в левой части `JOIN`, справа от которого он находится.

Когда элемент `FROM` содержит ссылки `LATERAL`, то запрос выполняется следующим образом: сначала вычисляется значения всех столбцов о которых зависит производная таблица с ключевым словом `LATERAL`, затем вычисляется сама производная таблица с `LATERAL` для каждой полученной записи. Результирующие строки полученные из производной таблицы с `LATERAL` соединяются со строками из которых они получены.

В качестве соединений допускается следующие `CROSS JOIN` и `LEFT OUTER JOIN`. Внутреннее соединение также допустимо, но не рекомендуется, поскольку могут возникнуть проблемы при вычислении условия соединения потоков.

Примеры соединений с боковыми производными таблицами:

```
select c.name, ox.order_date as last_order, ox.number
from customer c
left join LATERAL (select first 1 o.order_date, o.number
                   from orders o
                   where o.id_customer = c.id
                   order by o.ORDER_DATE desc
                  ) as ox on true;
```

```
select dt.population, dt.city_name, c.country_name
from (select distinct country_name from cities) AS c
cross join LATERAL (select first 1 city_name, population
                    from cities
                    where cities.country_name = c.country_name
                    order by population desc
                   ) AS dt;
```

```
select salespeople.name, max_sale.amount, customer_of_max_sale.customer_name
from salespeople,
     LATERAL (select max(amount) as amount
              from all_sales
              where all_sales.salesperson_id = salespeople.id
             ) as max_sale,
     LATERAL (select customer_name
              from all_sales
              where all_sales.salesperson_id = salespeople.id
                    and all_sales.amount = max_sale.amount
             ) as customer_of_max_sale;
```

11.1.7 WHERE

Предложение WHERE ограничивает возвращаемые строки только теми, которые соответствуют указанному условию. Условие, следующее за ключевым словом WHERE, может быть как простой проверкой (например, «AMOUNT = 3»), так и многоуровневым сложным выражением, содержащим подвыборки, предикаты, вызовы функций, математические и логические операторы, контекстные переменные и многое другое.

Листинг 11.13. Синтаксис предложения WHERE

```
SELECT ...
FROM ...
[WHERE <условие выборки>]
```

Условие выборки — это логическое выражение возвращающее TRUE, FALSE и возможно UNKNOWN. Только те строки, для которых условие поиска истинно будут включены в результирующий набор. Будьте осторожны с возможными получаемыми значениями NULL: если вы отрицаете выражение, дающее NULL с помощью NOT, то результат такого выражения все равно будет NULL и строка не пройдёт.

```
SELECT name
FROM wrestlers
WHERE region = 'Europe'
      AND weight > ALL (SELECT weight FROM shot_putters
                        WHERE region = 'Africa');
```

```
SELECT SUM(population)
FROM towns
WHERE name LIKE '%dam'
      AND province CONTAINING 'land';
```

В DSQL и ESQL выражение поиска может содержать параметры. Это полезно, если запрос должен быть повторен несколько раз с разными значениями входных параметров. В строке SQL-запроса, передаваемого на сервер, вопросительные знаки используются как обозначения для параметров. Эти вопросительные знаки называются позиционными параметрами, поскольку их можно различить только по их положению в строке. Библиотеки доступа часто поддерживают именованные параметры в виде :id, :amount, :a и т.д. Это более удобно для пользователя, библиотека заботится о преобразовании именованных параметров в позиционные параметры, прежде чем передать запрос на сервер.

Условие поиска может также содержать имена локальных (PSQL) или хостовых (ESQL) переменных, перед которыми ставится двоеточие.

```
SELECT name, address, phone
FROM stores
WHERE city = ? AND class = ?
```

```
SELECT *
FROM pants
WHERE model = :model AND size = :size AND color = :col;
```

Обратите внимание, что запросы, подобные этим, не могут быть выполнены немедленно, они должны быть предварительно подготовлены. После того как параметризованный запрос был подготовлен, пользователь (или вызывающий код) может подставить значения параметров и выполнять его многократно, подставляя перед каждым вызовом новые значения параметров.

11.1.8 GROUP BY

Предложение GROUP BY позволяет сгруппировать полученные в результате выполнения оператора SELECT строки. Оно соединяет записи, имеющие одинаковую комбинацию значений полей, указанных в его списке, в одну запись. Агрегатные функции в списке выбора применяются к каждой группе индивидуально, а не для всего набора в целом. Предложение может быть использовано в том случае, если в списке выбора присутствуют как имена столбцов, так и агрегатные функции AVG, COUNT, SUM, MIN, MAX и др.. При этом все столбцы, не являющиеся параметрами агрегатных функций, обязательно должны присутствовать в предложении GROUP BY. Это основное правило группирования.

Листинг 11.14. Синтаксис предложения GROUP BY

```
SELECT
      <список выборки>
FROM ...
GROUP BY <список группировки> [, <список группировки>];
[HAVING <условие выборки>]

<список группировки> ::=
  <неагрегирующее выражение в select>
| <неагрегирующее выражение>

<неагрегирующее выражение> ::=
  <копия выражения>
| <псевдоним столбца>
| <позиция столбца>
```

Здесь в качестве <списка группировки> может выступать:

- Дословная копия выражения из списка выбора, не содержащего агрегатной функции.
- Псевдоним выражения (столбца) из списка выбора, не содержащего агрегатной функции.

- Номер позиции выражения (столбца) из списка выбора, не содержащего агрегатной функции. Позиция представляет собой целое число от 1 до до количества столбцов в списке `SELECT`.
- Столбцы исходной таблицы, которые не включены в список выборки `SELECT`, или неагрегатные выражения, основанные на таких столбцах. Добавление таких столбцов может дополнительно разбить группы. Но так как эти столбцы не в списке выборки `SELECT`, вы не можете сказать, какому значению столбца соответствует значение агрегированной строки. Таким образом, если вы заинтересованы в этой информации, вы так же должны включить этот столбец или выражение в список выборки `SELECT`, что возвращает вас к правилу "каждый не агрегированный столбце в списке выборки `SELECT` должен быть включён в список группировки `GROUP BY`";
- Выражения, которые не зависят от данных из основного набора, т.е. константы, контекстные переменные, некоррелированные подзапросы, возвращающие единственное значение и т.д. Это упоминается только для полноты картины, т.к. добавление этих элементов является абсолютно бессмысленным, поскольку они не повлияют на группировку вообще. "Безвредные, но бесполезные" элементы так же могут фигурировать в списке выбора `SELECT` без их копирования в список группировки `GROUP BY`.

Если вы группируете по позиции столбца или алиасу, то выражение соответствующее этой позиции (алиасу) будет скопировано из списка выборки `SELECT`. Это касается и подзапросов, таким образом, подзапрос будет выполняться, по крайней мере, два раза.

Если список выборки содержит только агрегатные столбцы или столбцы, значения которых не зависит от отдельных строк основного множества, то предложение `GROUP BY` необязательно. Когда предложение `GROUP BY` опущено, результирующее множество будет состоять из одной строки (при условии, что хотя бы один агрегатный столбец присутствует).

```
SELECT COUNT(*), AVG(age), current_date
FROM students
WHERE sex = 'M';

COUNT  AVG  CURRENT_DATE
=====  ===  ============
157    11   11.03.2032
```

Если в списке выборки содержатся как агрегатные столбцы, так и столбцы, чьи значения зависит от выбираемых строк, то предложение `GROUP BY` становится обязательным.

```
SELECT
    class,
    sex,
    boarding_type,
    COUNT(*) AS anumber,
    AVG(age) AS avg_age
FROM students
GROUP BY class, sex, boarding_type;

CLASS SEX BOARDING_TYPE ANUMBER AVG_AGE
===== === ============= ======= ======
8A    F   BOARDING      2       12.0
8A    F   DAY           1       13.0
8A    M   DAY           3       14.0
8B    F   BOARDING      2       13.0
8B    M   BOARDING      1       14.0
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
  8B    M   DAY           1       14.0
```

Если при наличии предложения `GROUP BY`, в предложении `SELECT` отсутствуют агрегатные функции, то запрос просто вернет по одной строке из каждой группы.

HAVING

Необязательное предложение `HAVING` определяет дополнительные условия поиска (условия выборки строк таблицы/таблиц) для использования в `GROUP BY`. Предложение `HAVING` может использоваться только вместе с предложением `GROUP BY`.

Предложение `HAVING` вместе с предложением `WHERE`, предложениями `FIRST`, `SKIP`, `ORDER BY` и `ROWS` сокращает количество отобранных строк в результирующем выходном наборе данных.

Условие выборки в предложении `HAVING` может ссылаться на:

- Любой агрегированный столбец в списке выбора `SELECT`. Это наиболее широко используемый вариант.

```
SELECT
   class,
   COUNT(*) AS anumber,
   AVG(age) AS avg_age
FROM students
WHERE sex = 'M'
GROUP BY class
HAVING COUNT(*) >= 2;
```

- Любое агрегированное выражение, которое не находится в списке выбора `SELECT`, но допустимо в контексте запроса.

```
SELECT
   class,
   COUNT(*) AS anumber,
   AVG(age) AS avg_age
FROM students
WHERE sex = 'M'
GROUP BY class
HAVING MAX(age) - MIN(age) > 0.9;
```

- Любой столбец в списке `GROUP BY`. Однако более эффективно фильтровать неагрегированные данные на более ранней стадии в предложении `WHERE`.

```
SELECT
   class,
   COUNT(*) AS anumber,
   AVG(age) AS avg_age
FROM students
WHERE sex = 'M'
GROUP BY class
HAVING class STARTING WITH '8';
```

```
SELECT
   class,
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
    COUNT(*) AS anumber,
    AVG(age) AS avg_age
FROM students
WHERE sex = 'M' AND class STARTING WITH '8'
GROUP BY class;
```

- Любое выражение, значение которого не зависит от содержимого набора данных (например, константа или контекстная переменная). Это допустимо, но совершенно бессмысленно, потому что условие, не имеющее никакого отношения к самому набору данных, либо подавит весь набор, либо оставит его не тронутым.

Предложение HAVING не может содержать:

- Не агрегированные выражения столбца, которые не находятся в списке GROUP BY.
- Позицию столбца. Целое число в предложении HAVING – просто целое число.
- Псевдонимы столбца – даже, если они появляются в предложении GROUP BY.

11.1.9 WINDOW

Предложение WINDOW предназначено для определения именованных окон, которые используются оконными функциями. Поскольку выражение окна может быть довольно сложным и может использоваться многократно, такая функциональность бывает полезной.

Листинг 11.15. Синтаксис предложения WINDOW

```
SELECT ...
FROM ...
...
WINDOW <спецификация окна> [, <спецификация окна>] ...

<спецификация окна> ::= <имя окна> AS ([<имя окна>] [<выражение секционирования>]
                                       [<выражение сортировки>] [<рамка окна>])

<выражение секционирования> ::= PARTITION BY <выражение> [, <выражение> ...]

<выражение сортировки> ::= ORDER BY <выражение> [ASC | DESC] [NULLS {FIRST | LAST}]
                                  [, <выражение> [ASC | DESC] [NULLS {FIRST | LAST}]...
]

<рамка окна> ::= {ROWS | RANGE} { <начало окна> | <окно между значениями>}

<начало окна> ::= UNBOUNDED PRECEDING | <выражение> PRECEDING | CURRENT ROW

<окно между значениями> ::= BETWEEN { UNBOUNDED PRECEDING | <выражение> PRECEDING |
                                       <выражение> FOLLOWING | CURRENT ROW }
                                AND { UNBOUNDED FOLLOWING | <выражение> PRECEDING |
                                       <выражение > FOLLOWING | CURRENT ROW }
```

Имя окна может быть использовано в предложении OVER для ссылки на определение окна, кроме того оно может быть использовано в качестве базового окна для другого именованного или встроенного (в предложении OVER) окна. Окна с рамкой (с предложениями RANGE и ROWS) не могут быть использованы в качестве базового окна (но могут быть использованы в предложении OVER <имя окна>). Окно, которое использует ссылку на базовое окно, не может иметь предложение PARTITION BY и не может переопределять сортировку с помощью предложения ORDER BY.

Пример.

Приведен пример использования именованных окон (см. таблицу `employee` из тестовой БД `employee.fdb`).

```
SELECT
   emp_no,
   dept_no,
   salary,
   count(*) OVER w1,
   first_value(salary) OVER w2,
   last_value(salary) OVER w2,
   sum(salary) over (w2 ROWS BETWEEN CURRENT ROW AND 1 FOLLOWING) AS s
FROM employee
WINDOW w1 AS (PARTITION BY DEPT_NO),
       w2 AS (w1 ORDER BY salary)
ORDER BY dept_no, salary;
```

11.1.10 UNION

Предложение `UNION` позволяет объединить выбранный основным оператором `SELECT` набор данных с другим набором данных, имеющим точно такую же структуру, тем самым увеличивая общее количество строк, но не столбцов. Наборы данных, принимающие участие в `UNION`, должны иметь одинаковое количество столбцов. Однако столбцы в соответствующих позициях не обязаны иметь один и тот же тип данных, они могут быть абсолютно не связанными.

Не следует путать объединение таблиц, `UNION`, с соединением таблиц, `JOIN`. Синтаксис предложения объединения `UNION` следующий :

Листинг 11.16. Синтаксис предложения `UNION`

```
SELECT ...
FROM ...
...
UNION [DISTINCT | ALL] <выборка SELECT>
[UNION [DISTINCT | ALL] <выборка SELECT> ...]
```

Ключевое слово `ALL` означает, что в выходном наборе данных могут присутствовать дубликаты строк. При отсутствии этого ключевого слова или при задании `DISTINCT` дубликаты строк не помещаются в выходной набор данных. Отсутствие дубликатов строк в выходном наборе данных предполагается по умолчанию.

Объединения получают имена столбцов из первого запроса на выборку. Если вы хотите дать псевдонимы объединяемым столбцам, то делайте это для списка столбцов в самом верхнем запросе на выборку. Псевдонимы в других участвующих в объединении выборках разрешены и могут быть даже полезными, но они не будут распространяться на уровне объединения.

Если объединение имеет предложение `ORDER BY`, то единственно возможными элементами сортировки являются целочисленные литералы, указывающие на позиции столбцов, необязательно сопровождаемые `ASC/DESC` и/или `NULLS FIRST/LAST` директивами. Это так же означает, что вы не можете упорядочить объединение ничем, что не является столбцом объединения. Однако вы можете завернуть его в производную таблицу, которая даст вам все обычные параметры сортировки.

Объединения позволены в подзапросах любого вида и могут сами содержать подзапросы. Они также могут содержать соединения (`joins`) и могут принимать участие в соединениях, если завёрнуты в производную таблицу.

Пример 1.

Этот запрос представляет информацию из различных музыкальных коллекций в одном наборе данных с помощью объединений:

```
SELECT id, title, artist, len, 'CD' AS medium
FROM cds
UNION
     SELECT id, title, artist, len, 'LP'
     FROM records
UNION
     SELECT id, title, artist, len, 'MC'
     FROM cassettes
ORDER BY 3, 2;   -- artist, title
```

Если `id, title, artist` и `length` – единственные поля во всех участвующих таблицах, то запрос может быть записан так:

```
SELECT c.*, 'CD' AS medium
FROM cds c
UNION
     SELECT r.*, 'LP'
     FROM records r
UNION
     SELECT c.*, 'MC'
     FROM cassettes c
ORDER BY 3, 2;   -- artist, title
```

Квалификация "звёзд" необходима здесь, потому что они не являются единственным элементом в списке столбцов. Заметьте, что псевдонимы `"c"` в первой и третьей выборке не кусают друг друга. Они не имеют контекста объединения, а лишь применяются к отдельным запросам на выборку.

Пример 2.

Следующий запрос получает имена и телефонные номера переводчиков и корректоров. Те переводчики, которые также работают корректорами, будут отображены только один раз в результирующем наборе, если имена и номера их телефонов одинаковые в обеих таблицах. Тот же результат может быть получен без ключевого слова `DISTINCT`. Если вместо ключевого слова `DISTINCT`, будет указано ключевое слово `ALL`, эти люди будут отображены дважды.

```
SELECT name, phone
FROM translators
UNION DISTINCT
     SELECT name, telephone
     FROM proofreaders;
```

Пример 3.

Пример использования `UNION` в подзапросе:

```
SELECT name, phone, hourly_rate
FROM clowns
WHERE hourly_rate < ALL
     (SELECT hourly_rate FROM jugglers
      UNION
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
      SELECT hourly_rate FROM acrobats)
ORDER BY hourly_rate;
```

11.1.11 PLAN

При обработке оператора выборки данных SELECT оптимизатор сервера базы данных строит план, который определяет, в каком порядке и с использованием каких механизмов (при помощи индексов или при последовательном переборе с возможным дополнительным упорядочиванием) будут выбираться данные из таблиц для получения выходного набора данных. Как правило, такой план является не худшим решением. При этом у вас есть возможность задать свой план поиска, основываясь на каких-то характеристиках базы данных, например, на размерах различных индексов, количестве строк в таблицах и др. В некоторых случаях грамотно составленный собственный план может повысить производительность системы при выборке данных.

Утилита ISQL имеет возможность отобразить пользователю план извлечения данных с помощью команды SET PLAN ON. Для того, чтобы просто изучить план запроса (без выполнения самого запроса) необходимо ввести команду SET PLANONLY ON. Более подробный план можно получить при включении расширенного плана с помощью команды SET EXPLAIN ON.

Синтаксис для задания плана выборки представлен в листинге 11.17.

Листинг 11.17. Синтаксис предложения PLAN

```
SELECT ...
FROM ...
...
PLAN <выражение для плана поиска>

<выражение для плана поиска> ::=
              (<элемент плана> [,<элемент плана> ...])
              | SORT (<элемент плана>)
              | JOIN (<элемент плана> [,<элемент плана> ...])
              | [SORT] MERGE (<элемент плана> [,<элемент плана> ...])
              | HASH (<элемент плана> [,<элемент плана> ...])

<элемент плана> ::= <основной элемент> | <выражение для плана поиска>

<основной элемент> ::=
            { <имя/псевдоним таблицы> | <имя представления> }
            { NATURAL
            | INDEX (<имя индекса> [, <имя индекса> ...])
            | ORDER <имя индекса> [ INDEX (<имя индекса> [, <имя индекса> ...]) ] }
```

В этом синтаксисе можно видеть, что выражение для плана поиска допускает вложенные конструкции. Синтаксис для плана получается довольно богатым по своим возможностям.

Если выполняется запрос к нескольким таблицам, то на основании плана осуществляется выборка данных из каждой таблицы. Результатом одной такой выборки является промежуточный набор данных, который иногда называют потоком (stream). На следующем этапе выполняется соединение (JOIN) или объединение, или же слияние (MERGE) промежуточных наборов данных (потоков) в один результирующий набор данных, который и является результатом выполнения оператора SELECT.

В выражении плана в самом начале может присутствовать тип соединения. Используются типы соединения JOIN и MERGE.

JOIN — тип соединения по умолчанию. В этом случае с левым промежуточным набором данных

соединяются строки правого набора данных.

MERGE означает, что сливаются, объединяются два промежуточных набора данных — к левому промежуточному набору данных присоединяются строки правого набора данных. Ключевое слово SORT требует предварительной сортировки обоих наборов данных.

В элементе плана присутствует имя или псевдоним таблицы. Если для соответствующей таблицы был задан псевдоним, то и в элементе плана может присутствовать только псевдоним, но не имя таблицы.

После имени или псевдонима таблицы задаются ключевые слова NATURAL, INDEX или ORDER.

NATURAL (значение по умолчанию) означает, что все строки таблицы просматриваются последовательно страница за страницей вне какого-нибудь порядка и без использования каких-либо индексов.

Ключевое слово INDEX и следующий за ним в скобках список имен индексов данной таблицы задают использование указанных индексов для проверки условий соединения в запросе.

Ключевое слово ORDER указывает, что строки промежуточного набора данных должны быть упорядочены с использованием заданного индекса или соответствующего ограничения первичного, уникального или внешнего ключа.

Реализация действий по поиску данных на основании плана выполняется слева направо. При этом действия в круглых скобках выполняются, как обычно, в первую очередь.

В сложных запросах используются вложенные выражения для плана поиска. Соответствующие примеры планов будут рассмотрены далее.

Примеры простых планов

Здесь будут рассмотрены планы, которые строит оптимизатор запросов при обработке оператора выборки данных SELECT.

Далее в листингах будут приведены примеры запросов и сразу под ними планы в обычной и EXPLAIN форме (см. таблицы тестовой базы данных employee).

Пример 1.

Выбираются все столбцы всех строк таблицы JOB:

```
SELECT * FROM JOB;

PLAN (JOB NATURAL)

Select Expression
    -> Table "JOB" Full Scan
```

Пример 2.

Если есть предложение WHERE, то указывается индекс, который будет использоваться при нахождении совпадений (если такой существует):

```
SELECT * FROM JOB
WHERE JOB_COUNTRY = 'JOB_COUNTRY';

PLAN (JOB INDEX (RDB$FOREIGN3))

Select Expression
    -> Filter
        -> Table "JOB" Access By ID
            -> Bitmap
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
            -> Index "RDB$FOREIGN3" Range Scan (full match)
```

Пример 3.

В следующем запросе задается упорядоченность по столбцу, являющемуся первичным ключом таблицы.

```
SELECT * FROM JOB
ORDER BY JOB_CODE;

PLAN (JOB ORDER RDB$PRIMARY2)

Select Expression
   -> Table "JOB" Access By ID
      -> Index "RDB$PRIMARY2" Full Scan
```

В элементе плана задается упорядочение строк промежуточного набора данных с использованием индекса RDB$PRIMARY2, построенного системой автоматически для поддержания значений первичного ключа таблицы JOB.

Эквивалентным этому запросу будет запрос, содержащий такой план:

```
SELECT *
FROM JOB
PLAN SORT (JOB INDEX (RDB$PRIMARY2))
ORDER BY JOB_CODE;
```

Пример 4.

Для запроса, содержащего упорядочение по столбцу, не являющемуся ключевым и не входящему в состав какого-нибудь индекса будет построен план, в котором сразу после ключевого слова PLAN указывается упорядочение полученного набора данных.

```
SELECT * FROM JOB
ORDER BY JOB_TITLE;

PLAN SORT (JOB NATURAL)

Select Expression
   -> Sort (record length: 126, key length: 32)
      -> Table "JOB" Full Scan
```

В отличие от предыдущего примера, где из базы данных сразу выбирался упорядоченный набор данных, здесь выборка данных будет происходить в два этапа. В начале в результате последовательного (NATURAL) перебора строк таблицы будет сформирован промежуточный набор данных. Затем этот набор данных будет отсортирован в соответствии с заданными условиями упорядочения, указанными в предложении ORDER BY.

Пример 5.

В следующем запросе выбираются строки из таблицы JOB. Результат упорядочивается по столбцу "Страна работы", который входит в состав первичного ключа таблицы и в то же время является внешним ключом, ссылающимся на первичный ключ родительской таблицы — справочника стран COUNTRY. План задает выборку строк в порядке, указанном в индексе, который был автоматически создан системой для внешнего ключа (RDB$FOREIGN3).

```
SELECT * FROM JOB
ORDER BY job_country;

PLAN (JOB ORDER RDB$FOREIGN3)

Select Expression
   -> Table "JOB" Access By ID
      -> Index "RDB$FOREIGN3" Full Scan
```

Оптимизатор, естественно, выбирает упорядочение отыскиваемых строк на основании внешнего ключа.

Пример 6.

Если в операторе SELECT при выборке вакансий в предложении ORDER BY указать столбцы, входящие в состав первичного ключа (код вакансии, должность и название страны) в правильном порядке, то оптимизатор выберет упорядочение строк набора данных по первичному ключу.

```
SELECT * FROM JOB
ORDER BY job_code, job_grade, job_country;

PLAN (JOB ORDER RDB$PRIMARY2)

Select Expression
   -> Table "JOB" Access By ID
      -> Index "RDB$PRIMARY2" Full Scan
```

Пример 7.

Однако если в списке упорядочения переставить местами столбцы или изменить направление упорядочения на убывающее, то в такой таблице не найдется подходящего индекса.

```
SELECT * FROM JOB
ORDER BY job_grade, job_code, job_country;

PLAN SORT (JOB NATURAL)

Select Expression
   -> Sort (record length: 136, key length: 44)
      -> Table "JOB" Full Scan
```

Здесь последовательно будут выбраны соответствующие строки, а затем на втором этапе промежуточный набор данных упорядочивается нужным образом.

При создании первичного ключа для таблицы вакансий можно указать предложение USING, в котором задать упорядочение индекса по убыванию значений столбцов, входящих в его состав. В этом случае выборка данных с использованием предыдущего оператора будет проходить много быстрее.

Пример 8.

В следующем запросе данные выбираются также из таблицы вакансий, однако здесь присутствует предложение WHERE, задающее выборку не всех строк, а только тех, которые удовлетворяют указанному условию. В условии выборки используется столбец, являющийся внешним ключом таблицы. При этом никакая упорядоченность результата не задается. Оптимизатор построит план, в котором используется индекс, автоматически построенный системой для ограничения внешнего ключа.

```
SELECT * FROM JOB
WHERE job_country = 'USA';

PLAN (JOB INDEX (RDB$FOREIGN3))

Select Expression
   -> Filter
      -> Table "JOB" Access By ID
         -> Bitmap
            -> Index "RDB$FOREIGN3" Range Scan (full match)
```

Использование индекса внешнего ключа в данном случае позволяет резко ускорить выборку релевантных данных.

Пример 9.

В предыдущий запрос можно ввести требование упорядоченности результирующего набора данных по столбцам, входящим в состав первичного ключа. Созданный оптимизатором план теперь будет включать использование индекса первичного ключа для упорядочения строк набора данных на основании предложения ORDER BY.

```
SELECT * FROM JOB
WHERE job_code = 'SRep'
ORDER BY job_code, job_grade, job_country;

PLAN (JOB ORDER RDB$PRIMARY2)

Select Expression
   -> Filter
      -> Table "JOB" Access By ID
            -> Index "RDB$PRIMARY2" Range Scan (partial match: 1/3)
```

Примеры составных планов

Пример 1.

При выполнении более сложного запроса, включающего объединения (UNION) трех таблиц, оптимизатор создаст три плана для каждого оператора SELECT.

```
SELECT
    COD AS "Внутренний код",
    CODPEOPLE AS "Код человека",
    DATEADMIN AS "Дата",
    CODADMIN AS "Код нарушения/награды",
    'Нарушение' AS "Вид"
FROM PEOPLEADMIN
UNION
  SELECT
     COD AS "Внутренний код",
     CODPEOPLE AS "Код человека",
     DATESPEC AS "Дата",
     CODREW AS "Код нарушения/награды",
     'Награда' AS "Вид"
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
  FROM PEOPLEREW
UNION
  SELECT
     COD AS "Внутренний код",
     CODPEOPLE AS "Код человека",
     DATEADMIN AS "Дата",
     CODADMIN AS "Код нарушения/награды",
     'Нарушение' AS "Вид"
  FROM PEOPLEADMIN
ORDER BY 1;

PLAN SORT (PEOPLEADMIN NATURAL, PEOPLEREW NATURAL, PEOPLEADMIN NATURAL)

Select Expression
   -> Unique Sort (record length: 102, key length: 56)
      -> Union
         -> Table "PEOPLEADMIN" Full Scan
            -> Table "PEOPLEREW" Full Scan
               -> Table "PEOPLEADMIN" Full Scan
```

Для каждого из оператора выборки, заданных в предложении `UNION`, планы предусматривают последовательный перебор всех записей таблиц. Затем промежуточные наборы данных объединяются в один набор данных, строки которого сортируются по первому столбцу полученного набора данных (в планах не отражено).

Пример 2.

Следующий оператор `SELECT` выполняет двойное левое внешнее соединение таблицы с этой же таблицей.

```
SELECT
    PG.FULLNAME AS "Фамилия, имя, отчество",
    PM.NAME3 AS "Мать",
    PF.NAME3 AS "Отец"
FROM PEOPLE PG                   /* Главная таблица */
   LEFT OUTER JOIN PEOPLE PM     /* Мать */
      ON PG.CODMOTHER = PM.COD
   LEFT OUTER JOIN PEOPLE PF     /* Отец */
      ON PG.CODFATHER = PF.COD
ORDER BY PG.FULLNAME;

PLAN JOIN (SORT (JOIN (PG NATURAL, PM INDEX (RDB$PRIMARY1))), PF INDEX (RDB$PRIMARY1))

Select Expression
   -> Nested Loop Join (outer)
      -> Sort (record length: 270, key length: 108)
         -> Nested Loop Join (outer)
            -> Table "PEOPLE" as "PG" Full Scan
            -> Filter
               -> Table "PEOPLE" as "PM" Access By ID
                   -> Bitmap
                      -> Index "RDB$PRIMARY1" Unique Scan
      -> Filter
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
          -> Table "PEOPLE" as "PF" Access By ID
             -> Bitmap
                -> Index "RDB$PRIMARY1" Unique Scan
```

По этому плану видно, что строки главной таблицы выбираются последовательным перебором страниц базы данных. Для выборки соответствующих строк обеих соединяемых таблиц (это та же самая таблица) используется индекс, автоматически построенный системой для первичного ключа. Вначале к набору данных присоединяются данные из первой соединяемой таблицы (сведения о матери), затем к полученному набору данных присоединяются при использовании того же индекса данные из второй соединяемой таблицы (сведения об отце). После этого выполняется сортировка полученного набора данных.

Здесь в предложении `ORDER BY` для упорядочения результирующего набора данных используется вычисляемый столбец, для которого в базе данных не существует никакого индекса. Если же выполнить упорядочение набора данных по первичному ключу `COD` (в операторе `SELECT` возможно упорядочение и по столбцам, которые не присутствуют в списке выбора оператора), то план будет выглядеть несколько иначе.

```
SELECT
    PG.FULLNAME AS "Фамилия, имя, отчество",
    PM.NAME3 AS "Мать",
    PF.NAME3 AS "Отец"
FROM PEOPLE PG                   /* Главная таблица */
   LEFT OUTER JOIN PEOPLE PM     /* Мать */
      ON PG.CODMOTHER = PM.COD
   LEFT OUTER JOIN PEOPLE PF     /* Отец */
      ON PG.CODFATHER = PF.COD
ORDER BY PG.COD;

PLAN JOIN (JOIN (PG ORDER RDB$PRIMARY1, PM INDEX (RDB$PRIMARY1)), PF INDEX (RDB
$PRIMARY1))

Select Expression
   -> Nested Loop Join (outer)
      -> Nested Loop Join (outer)
         -> Table "PEOPLE" as "PG" Access By ID
            -> Index "RDB$PRIMARY1" Full Scan
         -> Filter
            -> Table "PEOPLE" as "PM" Access By ID
               -> Bitmap
                  -> Index "RDB$PRIMARY1" Unique Scan
      -> Filter
         -> Table "PEOPLE" as "PF" Access By ID
            -> Bitmap
               -> Index "RDB$PRIMARY1" Unique Scan
```

В этом случае не требуется выполнять дополнительную сортировку результирующего набора данных. Строки из главной таблицы выбираются уже в упорядоченном виде, поскольку используется индекс первичного ключа, по которому требуется отсортировать результирующий набор данных, как задано в предложении `ORDER BY`. Выборка соответствующих строк из соединяемых таблиц осуществляется точно так же, как и в предыдущем запросе.

Пример 3.

Выполнение довольно сложного запроса, который содержит как объединения (`UNION`), так и со-

единения (JOIN) нескольких таблиц, приведет к созданию также довольно сложного плана. Оптимизатор для этого запроса создаст три плана, определяющие порядок выборки данных из главных таблиц правонарушений и наград, которые указаны в предложениях FROM в операторах SELECT, и условия их соединения с таблицей людей (PEOPLE) в предложениях ON. Строки из всех таблиц выбираются последовательным перебором и для каждой строится HASH-таблица. Связь между главными и соединяемыми таблицам осуществляется через построенные HASH-таблицы..

```
SELECT *
FROM (SELECT
        P.FULLNAME AS "Сотрудник",
        DATEADMIN AS "Дата",
        CODADMIN AS "Код нарушения/награды"
      FROM PEOPLEADMIN
        INNER JOIN PEOPLE P
           ON PEOPLEADMIN.CODPEOPLE = P.COD
      UNION
         SELECT
           P.FULLNAME AS "Сотрудник",
           DATESPEC AS "Дата",
           CODREW AS "Код нарушения/награды"
         FROM PEOPLEREW
           INNER JOIN PEOPLE P
              ON PEOPLEREW.CODPEOPLE = P.COD
      UNION ALL
         SELECT
           P.FULLNAME AS "Сотрудник",
           DATEADMIN AS "Дата",
           CODADMIN AS "Код нарушения/награды"
         FROM PEOPLEADMIN
           INNER JOIN PEOPLE P
             ON PEOPLEADMIN.CODPEOPLE = P.COD)
ORDER BY "Сотрудник";

PLAN SORT (SORT (JOIN (P NATURAL, PEOPLEADMIN INDEX (RDB$FOREIGN3)), JOIN (P NATURAL,
PEOPLEREW INDEX (RDB$FOREIGN5))), JOIN (P NATURAL, PEOPLEADMIN INDEX (RDB$FOREIGN3)))

Select Expression
   -> Sort (record length: 194, key length: 108)
      -> Union
         -> Unique Sort (record length: 186, key length: 124)
            -> Union
              -> Nested Loop Join (inner)
                   -> Table "PEOPLE" as "P" Full Scan
               -> Filter
                -> Table "PEOPLEADMIN" Access By ID
                   -> Bitmap
                     -> Index "RDB$FOREIGN3" Range Scan (full match)
                       -> Nested Loop Join (inner)
                        -> Table "PEOPLE" as "P" Full Scan
         -> Filter
           -> Table "PEOPLEREW" Access By ID
             -> Bitmap
               -> Index "RDB$FOREIGN5" Range Scan (full match)
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
                -> Nested Loop Join (inner)
                 -> Table "PEOPLE" as "P" Full Scan
          -> Filter
            -> Table "PEOPLEADMIN" Access By ID
              -> Bitmap
                -> Index "RDB$FOREIGN3" Range Scan (full match)
```

Действия по выборке данных выполняются в соответствии с этими планами. В результате будет получено три промежуточных набора данных, которые затем объединяются в результирующий набор данных и сортируются по первому столбцу (упорядочивание результата в плане не отражено).

Пример 4.

В следующем запросе присутствуют агрегатные функции, выполняется полное внешнее соединение и группировка результата.

```
SELECT
   AVG(SALARY) AS "Среднее",
   MAX(SALARY) AS "Максимум",
   MIN(SALARY) AS "Минимум",
   COUNT(*) AS "Количество",
   O.NAME AS "Организация"
FROM STAFF S
   FULL OUTER JOIN FIRM O
      ON O.COD = S.CODORG
GROUP BY "Организация"
ORDER BY 5 COLLATE NONE;

PLAN SORT (SORT (JOIN (JOIN (O NATURAL, S INDEX (RDB$FOREIGN3)), JOIN (S NATURAL, O
INDEX (RDB$PRIMARY1)))))

Select Expression
   -> Sort (record length: 278, key length: 108)
      -> Aggregate
         -> Sort (record length: 166, key length: 108)
            -> Full Outer Join
               -> Nested Loop Join (outer)
                  -> Table "FIRM" as "O" Full Scan
                  -> Filter
                     -> Table "STAFF" as "S" Access By ID
                        -> Bitmap
                           -> Index "RDB$FOREIGN3" Range Scan (full match)
               -> Nested Loop Join (outer)
                  -> Table "STAFF" as "S" Full Scan
                  -> Filter
                     -> Table "FIRM" as "O" Access By ID
                        -> Bitmap
                           -> Index "RDB$PRIMARY1" Unique Scan
```

11.1.12 ORDER BY

Результат выборки данных при выполнении оператора SELECT по умолчанию никак не упорядочивается (фактически происходит упорядочение в хронологическом порядке помещения строк в таблицу операторами INSERT). Предложение ORDER BY позволяет задать при выборке данных из таблицы необходимый порядок. Синтаксис предложения представлен в листинге 11.18:

Листинг 11.18. Синтаксис предложения ORDER BY

```
SELECT ...
FROM ...
ORDER BY <определение сортировки> [, <определение сортировки> ... ]

<определение сортировки> ::=
   <выражение> [<определение порядка>] [<поведение для null>]

<определение порядка> ::=
  ASC  | ASCENDING
| DESC | DESCENDING

<поведение для null> ::=
  NULLS FIRST
| NULLS LAST
```

ORDER BY состоит из разделенного запятыми списка столбцов или выражений, по которым должен быть отсортирован полученный набор данных. Порядок сортировки может быть задан именем столбца, но только если этому столбцу ранее не был присвоен псевдоним в списке столбцов SELECT. Псевдоним должен быть использован, если он использовался в списке выбора. Порядковый номер столбца в списке столбцов SELECT, псевдоним, присвоенный столбцу в списке SELECT с помощью ключевого слова AS, или номер столбца в списке SELECT могут использоваться без ограничений.

Все три формы выражения столбцов для порядка сортировки можно смешивать в одном предложении ORDER BY. Например, один столбец в списке может быть указан по имени, а другой - по номеру.

Если вы сортируете по позиции столбца или его алиасу, то выражение соответствующее этой позиции (алиасу) будет скопировано из списка выборки SELECT. Это касается и подзапросов, таким образом, подзапрос будет выполняться, по крайней мере, два раза.

Если используется позиция столбца для указания порядка сортировки в запросах формата SELECT *, сервер расширяет звездочку до полного списка столбцов, чтобы определить столбцы для сортировки. Однако такая организация сортировки не считается хорошей практикой.

Ключевое слово ASCENDING задает упорядочение по возрастанию значений. Допустимо сокращение ASC. Применяется по умолчанию.

Ключевое слово DESCENDING задает упорядочение по убыванию значений. Допустимо сокращение DESC. В одном предложении упорядочение по одному столбцу может идти по возрастанию значений, а по другому — по убыванию.

Ключевое слово COLLATE в <выражении> определяет порядок сортировки, который следует применить к строковому столбцу, если нужен порядок, отличный от того, который был установлен для этого столбца (явно при описании столбца или по умолчанию, принятому для соответствующего набора символов).

Ключевое слово NULLS определяет, где в сортированном списке будут находиться пустые значения соответствующего столбца — в начале списка (FIRST) или в конце (LAST). По умолчанию принимается NULLS FIRST.

Пример.

Сортировка по псевдонимам столбцов:

```
SELECT
   RDB$CHARACTER_SET_ID AS CHARSET_ID,
   RDB$COLLATION_ID AS COLL_ID,
   RDB$COLLATION_NAME AS NAME
FROM RDB$COLLATIONS
ORDER BY CHARSET_ID, COLL_ID;
```

Сортировка частей UNION

Части выборок SELECT, участвующих в объединении UNION, не могут быть отсортированы с использованием предложения ORDER BY. Однако вы можете достичь желаемого результата с использованием производных таблиц или общих табличных выражений. Предложение ORDER BY, записанное последним в объединении, будет применено ко всей выборке в целом, а не к последней его части. Для объединений, единственно возможными элементами сортировки являются целочисленные литералы, указывающие на позиции столбцов, необязательно сопровождаемые ASC / DESC и/или NULLS FIRST / LAST директивами.

Пример.

Сортировка выборки полученной объединением выборок из двух запросов. Выборка сортируется по убыванию значений второго столбца с размещением NULL значений в конце списка и возрастанием значений первого столбца с размещением NULL значений в начале списка.

```
SELECT
   DOC_NUMBER, DOC_DATE
FROM PAYORDER
UNION ALL
   SELECT
      DOC_NUMBER, DOC_DATE
   FROM BUDGORDER
ORDER BY 2 DESC NULLS LAST, 1 ASC NULLS FIRST;
```

11.1.13 OPTIMIZE FOR

Предложение OPTIMIZE FOR позволяет задать необходимую стратегию оптимизации запросов, тем самым ускорить процесс выборки данных. Синтаксис предложения:

```
SELECT ...
   [WITH LOCK [SKIP LOCKED]]
   OPTIMIZE FOR {FIRST | ALL} ROWS
```

- FIRST - для запросов выбирается такой план доступа, который позволяет максимально быстро получить первые записи в выборке;
- ALL - для запросов выбирается такой план доступа, который позволяет максимально быстро получить все записи в выборке.

В отсутсвии предложения OPTIMIZE FOR при выполнении оператора SELECT будет использоваться стратегия оптимизации запросов, указанная в параметре конфигурационного файла OptimizeForFirstRows. Значением по умолчанию является default. Если при умолчательном значении параметра OptimizeForFirstRows в запросе присутствуют ключевые слова FIRST и/или SKIP

или же предложение ROWS, то будет использована стратегия оптимизации FIRST ROWS, несмотря на настройки файла конфигурации. Если же в конфигурационном файле указана стратегия ALL ROWS, то данные предложения не будут влиять на оптимизацию запросов. При стратегии ALL ROWS всегда применяется явный выбор данных и их сортировка.

Для примера рассмотрим запрос:

```
SELECT * FROM COUNTRY ORDER BY COUNTRY;
```

При стратегии FIRST ROWS для запроса, содержащего упорядочение по столбцу, будет построен следующий план:

```
PLAN (COUNTRY ORDER RDB$PRIMARY1)
```

Так выглядит план в расширенном варианте:

```
Select Expression
  -> Table "COUNTRY" Access By ID
     -> Index "RDB$PRIMARY1" Full Scan
```

А при стратегии ALL ROWS для этого же запроса оптимизатор построит другой план, в котором применяется упорядочивание данных полученного набора:

```
PLAN (COUNTRY ORDER RDB$PRIMARY1)
```

В расширенном варианте план выглядит так:

```
Select Expression
  -> Table "COUNTRY" Access By ID
     -> Index "RDB$PRIMARY1" Full Scan
```

11.1.14 ROWS

Предложение ROWS задает диапазон строк, которые попадут в результирующий набор данных из полученного в результате выполнения запроса набора данных. Синтаксис предложения:

Листинг 11.19. Синтаксис предложения ROWS

```
SELECT ...
FROM ...
ORDER BY ...
ROWS <m> [TO <n>]
```

Нумерация записей в наборе данных начинается с 1.

Предложение ROWS имеет смысл использовать, только если задано и предложение ORDER BY. ROWS может быть использовано без выражения ORDER BY, хотя это редко имеет смысл, за исключением случая, когда необходимо быстро взглянуть на данные таблицы – получаемые строки при этом будут чаще всего в случайном порядке.

В отличие от FIRST и SKIP, выражение ROWS принимает все типы целочисленных выражений в качестве аргумента – без скобок! Конечно, скобки могут требоваться для правильных вычислений внутри выражения, и вложенный запрос также должен быть обернут в скобки.

Если задан только один аргумент <m>:

- Вызов ROWS <m> приведёт к возвращению первых <m> записей из набора данных.

- Если <m> больше общего числа записей в возвращаемом наборе данных, то будет возвращён весь набор данных.
- Если <m> = 0, то будет возвращён пустой набор данных.
- Если <m> < 0, выдаётся ошибка

Пример.

Следующий запрос вернёт первые 10 имён из таблицы.

```
SELECT id, name FROM People
ORDER BY name ASC
ROWS 10;
```

Если заданы два аргумента <m> и <n>:

- Вызов ROWS <m> TO <n> приведёт к возвращению записей с <m> по <n> из набора данных.
- Если <m> больше общего количества строк в наборе данных и <n> >= <m>, то будет возвращён пустой набор данных.
- Если число <m> не превышает общего количества строк в наборе данных, а <n> превышает, то выборка ограничивается строками, начиная с <m> до конца набора данных.
- Если <m> < 1 и <n> < 1, то оператор SELECT выдаст ошибку.
- Если <n> = <m> -1, то будет возвращён пустой набор данных.
- Если <n> < <m> -1, то оператор SELECT выдаст ошибку.

Пример.

Этот запрос вернёт последние 10 записей.

```
SELECT id, name FROM People
ORDER BY name ASC
ROWS (SELECT COUNT(*) - 9 FROM People)
TO (SELECT COUNT(*) FROM People);
```

Соотношения между ключевыми словами FIRST и SKIP и предложением ROWS

Нельзя использовать ROWS вместе с FIRST/SKIP в одном и том же операторе SELECT, но можно использовать разный синтаксис в разных подзапросах. В сущности, ROWS заменяет собой нестандартные выражения FIRST и SKIP. Но не существует предложения ROWS, соответствующего тому случаю, когда заданы и ключевое слово FIRST, и ключевое слово SKIP.

Использование ROWS в UNION

При использовании ROWS с выражением UNION, он будет применяться к объединённому набору данных, и должен быть помещён после последнего SELECT.

При необходимости ограничить возвращаемые наборы данных одного или нескольких операторов SELECT внутри UNION, можно воспользоваться следующими вариантами:

- Использовать FIRST/SKIP в этих операторах SELECT. Необходимо помнить, что нельзя локально использовать выражение ORDER BY в SELECT внутри UNION – только глобально, ко всему суммарному набору данных.
- Преобразовать SELECT в производные таблицы с выражениями ROWS.

11.1.15 FETCH, OFFSET

Предложения FETCH и OFFSET являются SQL:2008 совместимым эквивалентом предложениям FIRST/SKIP и альтернативой предложению ROWS. Предложение OFFSET указывает, какое количество строк необходимо пропустить. Предложение FETCH указывает, какое количество строк необходимо получить. Синтаксис предложения:

Листинг 11.20. Синтаксис предложения OFFSET/FETCH

```
SELECT ...
FROM ...
[ORDER BY ...]
[OFFSET <значение1> {ROW | ROWS}]
[FETCH {FIRST | NEXT} [<значение2>] {ROW | ROWS} ONLY]
```

Здесь аргументами могут быть числовые литералы, SQL параметры (?) или PSQL параметры (:param).

Предложения OFFSET и FETCH могут применяться независимо уровня вложенности выражений запросов.

Предложения FETCH, OFFSET имеет смысл использовать, только если задано предложение ORDER BY.

Предложения OFFSET и/или FETCH не могут быть объединены с предложениями ROWS или FIRST/SKIP в одном выражении запроса.

В отличие от предложения ROWS, предложения OFFSET и FETCH допустимы только в операторе SELECT.

Пример 1.

Следующий запрос возвращает все строки кроме первых 10, упорядоченных по столбцу COL1:

```
SELECT * FROM T1
ORDER BY COL1
OFFSET 10 ROWS;
```

Пример 2.

В этом примере возвращается первые 10 строк, упорядоченных по столбцу COL1:

```
SELECT * FROM T1
ORDER BY COL1
FETCH FIRST 10 ROWS ONLY;
```

11.1.16 WITH LOCK

Опция WITH LOCK, обеспечивает возможность ограниченной явной пессимистической блокировки для осторожного использования в затронутых наборах строк крайне малой выборки (в идеале из одной строки) и при контроле из приложения.

> Требуется хорошее знание различных уровней изоляции и других параметров транзакций прежде чем использовать явную блокировку в вашем приложении.

Листинг 11.21. Синтаксис предложения WITH LOCK

```
SELECT ...
FROM ...
WITH LOCK [SKIP LOCKED]
```

При успешном выполнении предложения WITH LOCK будут заблокированы выбранные строки данных и таким образом запрещён доступ на их изменение в рамках других транзакций до момента завершения вашей транзакции.

Предложение WITH LOCK доступно только для выборки данных (SELECT) из одной таблицы. Предложение WITH LOCK нельзя использовать:

- в подзапросах;
- в запросах с объединением нескольких таблиц (JOIN);
- с оператором DISTINCT, предложением GROUP BY и при использовании любых агрегатных функций;
- при работе с представлениями;
- при выборке данных из селективных хранимых процедур;
- при работе с внешними таблицами.

Если предложение FOR UPDATE предшествует предложению WITH LOCK, то буферизация выборки не используется. Таким образом, блокировка применяется к каждой строке, одна за другой, по мере извлечения записей. Это делает возможным ситуацию, в которой успешная блокировка данных перестаёт работать при достижении в выборке строки, заблокированной другой транзакцией.

Сервер, в свою очередь, для каждой записи, подпадающей под явную блокировку, возвращает версию записи, которая является в настоящее время подтверждённой (актуальной), независимо от состояния базы данных, когда был выполнен оператор выборки данных, или исключение при попытке обновления заблокированной записи.

Ожидаемое поведение и сообщения о конфликте зависят от параметров транзакции, определённых в TPB (Transaction Parameters Block):

Таблица 11.3 — Влияние параметров TPB на явную блокировку

Режим TPB	Поведение
isc_tpb_consistency	Явные блокировки переопределяются неявными или явными блокировками табличного уровня и игнорируются.
isc_tpb_concurrency + isc_tpb_nowait	При подтверждении изменения записи в транзакции, стартовавшей после транзакции, запустившей явную блокировку, немедленно возникает исключение конфликта обновления.
isc_tpb_concurrency + isc_tpb_wait	При подтверждении изменения записи в транзакции, стартовавшей после транзакции, запустившей явную блокировку, немедленно возникает исключение конфликта обновления. Если в активной транзакции идёт редактирование записи (с использованием явной блокировки или нормальной оптимистической блокировкой записи), то транзакция, делающая попытку явной блокировки, ожидает окончания транзакции блокирования и, после её завершения, снова пытается получить блокировку записи. Это означает, что при изменении версии записи и подтверждении транзакции с блокировкой возникает исключение конфликта обновления.
isc_tpb_read_committed + isc_tpb_nowait	Если есть активная транзакция, редактирующая запись (с использованием явной блокировки или нормальной оптимистической блокировкой записи), то сразу же возникает исключение конфликта обновления.

(разрыв таблицы)

(разрыв таблицы)

Режим TPB	Поведение
`isc_tpb_read_committed +isc_tpb_wait`	Если в активной транзакции идёт редактирование записи (с использованием явной блокировки или нормальной оптимистической блокировкой записи), то транзакция, делающая попытку явной блокировки, ожидает окончания транзакции блокирования и, после её завершения, снова пытается получить блокировку записи. Для этого режима TPB никогда не возникает конфликта обновления

Как сервер работает с WITH LOCK

Попытка редактирования записи с помощью оператора `UPDATE`, заблокированной другой транзакцией, приводит к вызову исключения конфликта обновления или ожиданию завершения блокирующей транзакции – в зависимости от режима TPB. Поведение сервера здесь такое же, как если бы эта запись уже была изменена блокирующей транзакцией.

Нет никаких специальных кодов `gdscode`, возвращаемых для конфликтов обновления, связанных с пессимистической блокировкой.

Сервер гарантирует, что все записи, возвращённые явным оператором блокировки, фактически заблокированы и действительно соответствуют условиям поиска, заданным в операторе `WHERE`, если эти условия не зависят ни от каких других таблиц, не имеется операторов соединения, подзапросов и т.п. Это также гарантирует то, что строки, не попадающие под условия поиска, не будут заблокированы. Это не даёт гарантии, что нет строк, которые попадают под условия поиска, и не заблокированы.

Сервер блокирует строки по мере их выборки. Это имеет важные последствия, если блокируются сразу несколько строк. Многие методы доступа к базам данных по умолчанию используют для выборки данных пакеты из нескольких сотен строк (так называемый "буфер выборки"). Большинство компонентов доступа к данным не выделяют строки, содержащиеся в последнем принятом пакете, и для которых произошёл конфликт обновления.

Предостережения при использовании WITH LOCK

- Откат неявной или явной точки сохранения отменяет блокировку записей, которые изменялись в рамках её действий, но ожидающие окончания блокировки транзакции при этом не уведомляются. Приложения не должны зависеть от такого поведения, поскольку в будущем оно может быть изменено.
- Хотя явные блокировки могут использоваться для предотвращения и/или обработки необычных ошибок конфликтов обновления, объем ошибок обновления (`deadlock`) вырастет, если вы тщательно не разработаете свою стратегию блокировки и не будете ей строго управлять.
- Большинство приложений не требуют явной блокировки записей. Основными целями явной блокировки являются:
  1) предотвращение дорогостоящей обработки ошибок конфликта обновления в сильно загруженных приложениях
  2) для поддержания целостности объектов, отображаемых из реляционной базы данных в кластеризируемой среде.

  Если использование вами явной блокировки не подпадает под одну из этих двух категорий, то это является неправильным способом решения задач.
- Явная блокировка — это расширенная функция; не злоупотребляйте её использованием! В то время как явная блокировка может быть очень важной для веб-сайтов, обрабатывающих тысячи параллельных пишущих транзакций или для систем типа `ERP/CRM`, работающих в крупных корпорациях, большинство прикладных программ не требуют её использования.

11.1.17 INTO

В PSQL (хранимых процедурах, триггерах и др.) результаты выборки команды SELECT могут быть построчно загружены в локальные переменные (число, порядок и типы локальных переменных должны соответствовать полям SELECT). Часто такая загрузка – единственный способ что-то сделать с возвращаемыми значениями.

Листинг 11.22. Синтаксис предложения INTO

```
SELECT ...
FROM ...
INTO [:] <psql переменная> [, [:] <psql переменная> ...]
```

В PSQL двоеточие перед именами переменных является опциональным.

Простой оператор SELECT может быть использован в PSQL, только если он возвращает единственную строку. Для запросов, возвращающих несколько строк, PSQL предлагает использовать оператор FOR SELECT.

Также, PSQL поддерживает оператор DECLARE CURSOR, который связывает именованный курсор с определенной командой SELECT — и этот курсор впоследствии может быть использован для навигации по возвращаемому набору данных.

В PSQL выражение INTO должно появляться в самом конце команды SELECT.

Пример.

В PSQL, можно присвоить значения min_amt, avg_amt и max_amt заранее объявленным переменным или выходным параметрам:

```
CREATE PROCEDURE GetOrderStats
  RETURNS (min_amt DECIMAL(10, 2), avg_amt FLOAT, max_amt DECIMAL(10, 2))
AS
BEGIN
  SELECT
     MIN(amount),
     AVG( CAST(amount AS float) ),
     MAX(amount)
  FROM orders
  WHERE artno = 372218
  INTO min_amt, avg_amt, max_amt;
END!
```

11.1.18 Полный синтаксис SELECT

В предыдущих разделах использовались неполные или упрощенные фрагменты синтаксиса SELECT. Ниже приводится полный синтаксис.

Приведенный ниже синтаксис не включает синтаксис PSQL SELECT ... INTO, который по сути является: <спецификацией курсора> INTO <список переменных>.

Листинг 11.23. Полный синтаксис оператора SELECT

```
<спецификация курсора> ::=
   <запрос> [<предложение update>] [<предложение lock>]

<запрос> ::=
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
    [<предложение with>] <тело запроса> [<предложение order by>]
       [{ <предложение rows>
       | [<предложение offset>] [<предложение fetch>] }]

<предложение with> ::=
   WITH [RECURSIVE] <CTE> [, <CTE> ...]

<CTE> ::= <псевдоним CTE> [(<список столбцов CTE>)] AS (<оператор SELECT или UNION>)

<список столбцов> ::= <столбец> [, <столбец> ...]

<тело запроса> ::=
     <основа запроса>
   | <тело запроса> UNION [{ DISTINCT | ALL }] <другой набор данных>

<основа запроса> ::=
     <определение запроса>
   | (<тело запроса> [<предложение order by>]
   [<предложение offset>] [<предложение fetch>])

<определение запроса> ::=
   SELECT <ограничение> [{ ALL | DISTINCT }] <список выбора>
      FROM <источник> [, <источник> ...]
      [WHERE <условие поиска>]
      [GROUP BY <значение> [, <значение> ...]]
      [HAVING <условие поиска>]
      [WINDOW <определение окна> [, <определение окна> ...]]
      [PLAN <выражение>]

<ограничение> ::= [FIRST <выражение ограничения>] [SKIP <выражение ограничения>]

<выражение ограничения> ::=
  <целочисленный литерал>
| <параметр запроса>
| (<выражение>)

<список выбора> ::= * | <подсписок> [, <подсписок> ...]

<подсписок> ::=
  <спецификатор>.*
| <выражение> [[AS] <столбец>]

<источник> ::= <табличный источник> | <соединение>

<табличный источник> ::=
  <источник или CTE> [[AS] псевдоним]
| [LATERAL] <производная таблица> [ [AS] <псевдоним> [(<список столбцов>)]]
| <родительская соединённая таблица>

<источник или CTE> ::=
  <таблица или представление>
| CTE
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
| [имя пакета.]<имя процедуры> [(<аргументы процедуры>)]

<аргументы процедуры> ::= <значение> [, <значение> ...]

<производная таблица> ::=
   (<SELECT запрос>) [[AS] <псевдоним производной таблицы>]
   [(<псевдоним столбца> [, <псевдоним столбца>])]

<список столбцов> ::= <столбец> [, <столбец> ...]

<столбец> ::=
  <таблица или псевдоним>
| <представление>
| <CTE>
| <производная таблица>

<родительская соединённая таблица> ::=
  (<родительская соединённая таблица>)
| (<соединение>)

<соединение> ::=
  <перекрёстное соединение>
| <естественное соединение>
| <квалифицированное соединение>

<перекрёстное соединение> ::=
   <источник> CROSS JOIN <табличный источник>

<естественное соединение> ::=
   <источник> NATURAL [<тип соединения>] JOIN <табличный источник>

<тип соединения> ::= INNER | { LEFT | RIGHT | FULL } [OUTER]

<квалифицированное соединение> ::=
   <источник> [<тип соединения>] JOIN <табличный источник>
   { ON <условие соединения>
   | USING (<список столбцов>) }

<предложение order by> ::=
   ORDER BY <определение сортировки> [, <определение сортировки> ...]

<определение сортировки> ::=
   <выражение> [<определение порядка>] [<поведение для null>]

<определение порядка> ::=
  ASC  | ASCENDING
| DESC | DESCENDING

<поведение для null> ::=
  NULLS FIRST
| NULLS LAST
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
<окно> ::=
   <имя окна> AS (<определение окна>)

<определение окна> ::=
   [имя существующего окна]
   [<выражение секционирования>]
   [<предложение order by>]
   [<определение рамки>]

<выражение секционирования> ::=
   PARTITION BY <выражение> [, <выражение> ...]

<определение рамки> ::= { RANGE | ROWS } <границы рамки>

<границы рамки> ::=
     <начало рамки>
   | <рамка между>

<начало рамки> ::=
     UNBOUNDED PRECEDING
   | <выражение> PRECEDING
   | CURRENT ROW

<рамка между> ::=
   BETWEEN { UNBOUNDED PRECEDING | <выражение> PRECEDING
           | CURRENT ROW | <выражение> FOLLOWING }
   AND { <выражение> PRECEDING | CURRENT ROW
       | <выражение> FOLLOWING | UNBOUNDED FOLLOWING }

<предложение rows> ::= ROWS <значение от> [TO <значение до>]

<предложение offset> :: =
   OFFSET <выражение выборки> { ROW | ROWS }

<выражение выборки> ::=
     <целочисленный литерал>
   | <параметр запроса>

<предложение fetch> ::=
   [FETCH { FIRST | NEXT }
      [<выражение выборки>] { ROW | ROWS } ONLY]

<предложение WITH LOCK> ::= WITH LOCK [SKIP LOCKED]
```

11.2 INSERT

Добавление новых строк в обычную таблицу или в таблицу, лежащую в основе представления, осуществляется с помощью оператора `INSERT`. Его синтаксис представлен в листинге 11.24.

Листинг 11.24. Синтаксис оператора добавления данных INSERT

```
INSERT INTO {<имя таблицы> | <имя представления>}
[OVERRIDE {SYSTEM | USER} VALUE]
{ DEFAULT VALUES | [(<список столбцов>)] <источник значений>}
[RETURNING { <список возвращаемых значений> | *} [INTO <переменные>]]

<источник значений> ::=  VALUES (<значение>| DEFAULT [,<значение>| DEFAULT...])
                       | <поиск многих>

<список столбцов> ::= <имя столбца> [, <имя столбца> ...]

<список возвращаемых значений> ::= <имя столбца> [[AS] <алиас>] [, <имя столбца>
[[AS] <алиас>]...]

<переменные> ::= [:]<имя переменной> [, [:]<имя переменной> ...]

<значение> ::=  <литерал>
              | <выражение>
              | <встроенная функция>
              | <UDF> [(<параметр> [, <параметр> ...])]
              | <обращение к хранимой процедуре> [(<параметр> [, <параметр>]...)]
              | NEXT VALUE FOR <имя генератора>
              | (<выбор одного>) }
```

Поиск многих — оператор SELECT, возвращающий ноль или произвольное количество значений нескольких столбцов.

Добавлять данные в таблицу может ее владелец, пользователь SYSDBA, пользователь операционной системы root (Linux), trusted user (Windows), а также пользователь, которому предоставлено право добавлять данные в таблицу (в таблицы, базовые для представления) оператором GRANT INSERT — см. документ "Руководство администратора".

Можно добавить строку, для которой указаны конкретные значения столбцов; также строку, которая будет во всех столбцах содержать значения по умолчанию (предложение DEFAULT VALUES); или значения столбцов также могут быть получены из оператора SELECT, в этом случае может быть вставлено от нуля и более строк.

После имени таблицы или имени представления в скобках могут быть указаны имена столбцов, в которые будут помещаться значения. Если список столбцов не указан, то данные будут помещаться в том порядке, в котором столбцы присутствуют в таблице или в порядке описания столбцов базовой таблицы в представлении. Рекомендуется всегда указывать список столбцов. Во-первых, это избавит от ошибок в случае изменения порядка столбцов в таблице или в представлении. Во-вторых, это некоторый элемент документирования. Это особенно важно, если таблица или представление содержит достаточно большое количество столбцов.

```
INSERT INTO cars (make, model, byyear)
VALUES ('Ford', 'T', 1908);

INSERT INTO cars
VALUES ('Ford', 'T', 1908, 'USA', 850);
```

Если в списке столбцов оператора INSERT не указан какой-либо столбец, то ему будет присвоено значение по умолчанию (предложение DEFAULT в определении столбца). Если для такого столбца не

задано значение по умолчанию, то ему будет присвоено пустое значение NULL. Для столбца, входящего в состав первичного ключа, пустые значения недопустимы. Столбцы, для которых запрещено пустое значение, и которые не имеют значения по умолчанию, отличного от NULL, обязательно должны быть указаны в операторе INSERT с конкретным значением.

11.2.1 VALUES

В случае использования ключевого слова VALUES в списке, следующем за этим ключевым словом, должны быть представлены значения столбцов, помещаемых в строку таблицы. Весь список заключается в скобки, значения в списке отделяются друг от друга запятыми. Количество значений должно в точности совпадать с количеством имен столбцов, перечисленных в списке столбцов оператора, или всех существующих в таблице столбцов при отсутствии такого списка.

В списке VALUES вместо значения столбца можно использовать ключевое слово DEFAULT. В этом случае столбец получит значение по умолчанию, указанное при определении целевой таблицы. Если значение по умолчанию для столбца отсутствует, то столбец получит значение NULL.

```
CREATE TABLE cars (
   ID INTEGER GENERATED BY DEFAULT AS IDENTITY,
   BYYEAR SMALLINT DEFAULT 1990 NOT NULL,
   NAME VARCHAR(45),
   CONSTRAINT pk_cars PRIMARY KEY (ID) );

INSERT INTO cars (byyear, name)
VALUES (DEFAULT, 'Ford Focus' );
-- в столбец BYYEAR попадёт значение 1990

INSERT INTO cars (id, byyear, name)
VALUES (DEFAULT, 1996, 'Ford Mondeo');
-- в столбец id попадёт значение 2, как будто мы не указывали значение для id
```

Если ключевое слово DEFAULT указано для столбца, определенного как GENERATED BY DEFAULT AS IDENTITY, то столбец получит следующее значение идентификации, так как будто этот столбец не был указан в запросе вовсе.

11.2.2 Поиск многих

Поиск многих — оператор SELECT, возвращающий ноль или произвольное количество значений нескольких столбцов.

Данные для вставки в таблицу или представление можно получить из сколь угодно сложного оператора SELECT из какой-либо таблицы или группы таблиц, представления или хранимой процедуры выбора. Поиск многих позволяет выполнить именно "пакетное" добавление данных. В этом варианте в таблицу помещается столько строк, сколько вернет внутренний оператор SELECT. Только в этом случае оператор SELECT можно в операторе добавления данных не заключать в круглые скобки.

Для использования оператора SELECT в операторе добавления данных пользователь должен иметь соответствующие полномочия по выборке этих данных — быть владельцем всех таблиц, из которых осуществляется выборка данных, пользователем SYSDBA, пользователем root операционной системы (Linux), trusted user (Windows) или быть пользователем, которому предоставлено право выборки данных из всех этих таблиц оператором GRANT SELECT — см. документ "Руководство администратора".

Количество столбцов, возвращаемых оператором SELECT, должно в точности соответствовать количеству столбцов добавляемой строки. По этой причине имеет смысл в операторе SELECT явно задавать список выбора в виде списка имен столбцов, а не использовать символ "*", который определяет выбор всех столбцов таблицы. Список выбора может содержать не только имена столбцов, но и литералы.

```
INSERT INTO cars (make, model, byyear)
SELECT make, model, byyear
FROM new_cars;
```

Следует быть особенно осторожными при добавлении в таблицу строк, получаемых из той же самой таблицы, что присутствует в операторе SELECT, так как в подобных случаях можно получить бесконечно выполняемый оператор — когда вновь добавленные строки опять возвращаются в оператор добавления в результате выполнения оператора SELECT.

Пример.

Этот пример не является совсем правильным, потому что есть более простые способы выполнить соответствующие действия проще и за меньшее количество серверного времени и используемых ресурсов. Пусть, например, нужно переписать одни ошибочно записанные регионы страны Россия в другую страну, например, США. Можно выполнить следующий оператор добавления в таблицу регионов REGION:

```
INSERT INTO REGION (CODCOUNTRY, CODREGION, ...)
SELECT 'USA', CODREGION, ...
FROM REGION
WHERE (CODCOUNTRY = 'RUS' AND CODREGION = 'TX');
```

Здесь в списке выбора оператора SELECT первым указан литерал 'USA'. Значение этого литерала будет присвоено столбцу CODCOUNTRY для всех добавляемых строк.

После выполнения этого оператора следует удалить все переписанные строки из страны Россия, выполнив следующий оператор:

```
DELETE FROM REGION
WHERE (CODCOUNTRY = 'RUS' AND CODREGION = 'TX');
```

Подробнее об операторе удаления строк таблицы см. в конце этой главы. Более эффективный вариант этого действия см. в следующем разделе.

Подобное решение следует рассматривать только как иллюстрацию конкретного оператора. Если таблица регионов содержит подчиненные таблицы (таблицы, чьи внешние ключи ссылаются на данную таблицу, а их может быть более одной), то такая форма перемещения данных в некоторых случаях может привести к нарушениям целостности данных базы данных или к невыполнению соответствующих перемещений для подчиненных данных. Лучшим вариантом будет использование оператора UPDATE, правда только в том случае, если в описании внешних ключей всех дочерних таблиц использовались варианты ON DELETE CASCADE и ON UPDATE CASCADE или в базе данных существуют триггеры, которые выполняют соответствующие действия. См. главу 9, главу 20.

11.2.3 DEFAULT VALUES

Предложение DEFAULT VALUES позволяет вставлять записи без указания значений вообще. Это возможно, только если для каждого NOT NULL поля и полей, на которые наложены другие ограничения, или имеются допустимые объявленные значения по умолчанию, или эти значения устанавливаются в BEFORE INSERT триггере.

```
INSERT INTO journal
DEFAULT VALUES;
```

11.2.4 OVERRIDING

Значения столбцов идентификации (GENERATED BY DEFAULT AS IDENTITY) могут быть переопределены в операторах INSERT, UPDATE OR INSERT, MERGE. Для этого просто достаточно указать значение столбца в списке значений. Однако для столбцов определённых как GENERATED ALWAYS это недопустимо. Директива OVERRIDING SYSTEM VALUE позволяет заменить сгенерированное системой значение на значение указанное пользователем. Директива OVERRIDING SYSTEM VALUE вызовет ошибку, если в таблице нет столбцов идентификации или если они определены как GENERATED BY DEFAULT AS IDENTITY.

```
CREATE TABLE objects (
  id INT GENERATED ALWAYS AS IDENTITY,
  name CHAR(50));

INSERT INTO objects (id, name)
OVERRIDING SYSTEM VALUE
VALUES (11, 'Laptop' );   -- будет вставлено значение с кодом 11
```

Директива OVERRIDE USER VALUE выполняет обратную задачу, т.е. заменяет значение указанное пользователем на значение сгенерированное системой, если столбец идентификации определён как GENERATED BY DEFAULT AS IDENTITY. Директива OVERRIDING USER VALUE вызовет ошибку, если в таблице нет столбцов идентификации или если они определены как GENERATED ALWAYS AS IDENTITY.

```
CREATE TABLE objects (
  id INT GENERATED BY DEFAULT AS IDENTITY,
  name CHAR(50));

INSERT INTO objects (id, name)
OVERRIDING USER VALUE
VALUES (12, 'Laptop' );   -- значение 12 будет проигнорировано
```

11.2.5 RETURNING

Необязательное предложение RETURNING указывает, что оператор возвращает значения заданных столбцов одной добавляемой строки. Список столбцов возвращаемых значений не заключается в скобки. Если оператор помещает более одной строки в таблицу, то в этом случае возникнет ошибка базы данных. Ключевое слово INTO позволяет сохранить возвращенные значения во внутренних переменных триггера или хранимой процедуры. Здесь список также не заключается в скобки. Вариант INTO может использоваться только в PSQL.

В DSQL, оператор с RETURNING всегда возвращает только одну строку. Если указано предложение RETURNING, и обнаружено более одной совпадающей строки, выдаётся сообщение об ошибке.

Вместо списка столбцов вы можете указать звёздочку (*) для возврата всех значений столбцов таблицы. Возвращаемые значения содержат все изменения, произведённые в триггерах BEFORE.

Замечания:

- Предложение RETURNING поддерживается только для INSERT .. VALUES и одиночных INSERT .. SELECT.
- В DSQL оператор с использованием RETURNING всегда возвращает ровно одну строку. Если запись не была вставлена на самом деле, то все поля в возвращаемой строке будут меть значения NULL.
- В PSQL, если ни одна строка не вставлена, то ничего не возвращается и все целевые переменные сохраняют свои прежние значения.

11.3 UPDATE

Для изменения данных в столбцах существующих строк в обычной таблице или в таблице, являющейся базовой для изменяемого представления, используется оператор UPDATE. Оператор за одно обращение позволяет изменить данные во всех строках или только в части строк в таблице или в представлении. Его синтаксис представлен в листинге 11.25.

Листинг 11.25. Синтаксис оператора изменения данных UPDATE

```
UPDATE {<имя таблицы> | <имя представления>} [[AS] <псевдоним>]
SET <имя столбца> = <значение>|DEFAULT [,<имя столбца> = <значение>|DEFAULT...]
[WHERE { <условие поиска> | CURRENT OF <имя курсора>}]
[PLAN <план>]
[ORDER BY <упорядочиваемый элемент> [, <упорядочиваемый элемент> ... ]]
[ROWS <m> [TO <n>]]
[SKIP LOCKED]
[RETURNING { <возвращаемые значения> | [{OLD. | NEW.}]* }
   [INTO [:]<имя переменной> [, [:]<имя переменной> ...]] ];

<значение> ::= { <литерал>
               | <выражение>
               | <встроенная функция>
               | <UDF> [(<параметр> [, <параметр> ...])]
               | NEXT VALUE FOR <имя генератора>
               | (<выбор одного>) }

<упорядочиваемый элемент> ::=
   {<имя столбца>|<псевдоним столбца>|<номер столбца>|<произвольное выражение>}
   [COLLATE <порядок сортировки>]
   [ASC[ENDING] | DESC[ENDING]]
   [NULLS {FIRST | LAST}]

<возвращаемые значения> ::= <имя столбца> [[AS] <алиас>] [,<имя столбца> [[AS] <алиас>
]]...
```

Изменять данные в таблице может ее владелец, пользователь SYSDBA, пользователь операционной системы root (Linux), trusted user (Windows), а также пользователь, которому предоставлено право изменять отдельные (указанные в операторе) столбцы таблицы оператором GRANT UPDATE — см. документ "Руководство администратора". Если в данном операторе изменение ключевых столбцов (столбцов, входящих в состав первичного или уникального ключа) таблицы влечет автоматическое внесение изменений в строки дочерних таблиц, то и к этим таблицам пользователь должен иметь полномочия UPDATE.

Если вы назначаете псевдоним таблице или представлению, вы обязаны использовать псевдоним для уточнения столбцов таблицы.

11.3.1 SET

Предложение SET задает список выполняемых изменений: указывается имя изменяемого столбца и после знака равенства новое значение столбца.

Разрешено использовать имена столбцов в качестве значения. При этом использоваться будет всегда старое значение столбца, даже если присваивание этому столбцу уже произошло ранее в перечислении SET. Один столбец может быть использован только один раз в конструкции SET.

Пример.

Пусть имеются следующие данные в таблице SETTBL:

```
A            B
============ ============
1            0
2            0
```

После выполнения оператора:

```
UPDATE SETTBL
SET A = 5, B = A;
```

получим следующую таблицу:

```
A            B
============ ============
5            1
5            2
```

Обратите внимание, что старые значения (1 и 2) используются для обновления столбца B, даже после того как столбцу A были назначено новое значение (5).

В предложении SET вместо значения столбца можно использовать ключевое слово DEFAULT. В этом случае столбец получит значение по умолчанию, указанное при определении целевой таблицы. Если значение по умолчанию для столбца отсутствует, то столбец получит значение NULL.

```
UPDATE cars
SET BYYEAR = DEFAULT
WHERE ID = 1;
```

11.3.2 WHERE

Предложение WHERE ограничивает набор обновляемых записей заданным условием или текущей строкой именованного курсора, если указано предложение WHERE CURRENT OF (только в PSQL). Если это предложение не указано, то будут изменены все строки таблицы в том случае, если не было указано также предложение ORDER BY вместе с предложением ROWS. Для того чтобы иметь возможность использовать предложение WHERE в операторе UPDATE, пользователь должен также иметь полномочия выборки данных из этой же таблицы — GRANT SELECT.

```
UPDATE employees
SET salary = 2.5 * salary
WHERE title = 'CEO';
```

Подробнее об условиях поиска в предложении WHERE см. в разделе 11.1.

11.3.3 PLAN

В операторе изменения данных может быть использовано ключевое слово PLAN, задающее план выборки данных. Подробнее о планах выборки см. в разделе 11.1.11. Основным назначением этого предложения является определение порядка выборки строк из таблицы, для которых будут выполняться изменения. Такое предложение может ускорить (или замедлить при неправильном его применении) выборку подлежащих для корректировки записей. Как правило, сервер базы данных выбирает наибо-

лее подходящий план для поиска записей. При достаточном опыте работы с базой данных вы можете задавать свой собственный план, позволяющий ускорить работу системы.

11.3.4 ORDER BY и ROWS

Предложение ORDER BY позволяет задать порядок обновления записей. Это может иметь значение в некоторых случаях.

Листинг 11.26. Синтаксис предложения упорядочения данных ORDER BY

```
UPDATE ...
SET ...
ORDER BY <упорядочиваемый элемент> [, <упорядочиваемый элемент> ... ]
[ROWS <m> [TO <n>]]

<упорядочиваемый элемент> ::=
   {<имя столбца>|<псевдоним столбца>|<номер столбца>|<произвольное выражение>}
   [COLLATE <порядок сортировки>]
   [ASC[ENDING] | DESC[ENDING]]
   [NULLS {FIRST | LAST}]
```

В этом предложении перечисляются столбцы, по которым нужно упорядочить строки набора данных перед выполнением изменений. Здесь можно задавать только имена столбцов, а не их номера.

Ключевое слово ASCENDING задает упорядочение по возрастанию значений. Допустимо сокращение ASC. Применяется по умолчанию.

Ключевое слово DESCENDING задает упорядочение по убыванию значений. Допустимо сокращение DESC. В одном предложении упорядочение для одного столбца может идти по возрастанию значений, а для другого — по убыванию.

Ключевое слово COLLATE позволяет задать порядок сортировки строкового столбца, если нужен порядок, отличный от того, который был установлен для этого столбца (явно или по умолчанию). Допустимые порядки сортировки для различных наборов символов см. в Приложение Д.

Ключевое слово NULLS определяет, где в сортированном списке будут находиться пустые значения соответствующего столбца — в начале всего списка (FIRST) или в конце (LAST). По умолчанию принимается NULLS FIRST.

Для возможности использования предложения ORDER BY в операторе UPDATE пользователь должен также иметь полномочия выборки данных из соответствующей таблицы — GRANT SELECT.

Предложение ROWS имеет смысл только вместе с предложением ORDER BY. Однако его можно использовать отдельно.

При одном аргументе m, ROWS ограничивает обновление первыми m строками. Особенности:

- Если m > количества обрабатываемых строк, то обновляется весь набор строк;
- Если m = 0, ни одна строка не обновляется;
- Если m < 0, выдаётся ошибка.

При двух аргументах m и n, ROWS ограничивает обновление до строк от m до n включительно. Оба аргумента – целочисленные, и начинаются с 1. Особенности:

- Если m > количества обрабатываемых строк, ни одна строка не обновляется;
- Если n больше количества строк, то обновляются строки от m до конца набора;
- Если m < 1 или n < 1, выдаётся ошибка;
- Если n = m - 1, ни одна строка не обновляется;
- Если n < m - 1, выдаётся ошибка.

Пример.

Надбавка 20-ти сотрудникам с наименьшей зарплатой.

```
UPDATE employees
SET salary = salary + 50
ORDER BY salary ASC
ROWS 20;
```

В одном операторе могут быть указаны и предложение WHERE, и предложение ROWS. В этом случае сначала отбираются строки, соответствующие условию в предложении WHERE, затем они упорядочиваются на основании предложения ORDER BY, и, наконец, выполняются заданные изменения для тех строк, которые указаны в предложении ROWS.

11.3.5 RETURNING

Необязательное предложение RETURNING указывает, что оператор возвращает значения заданных столбцов обновляемой строки. В RETURNING могут включаться любые строки, необязательно только те, которые обновляются. Если оператор изменяет более одной строки таблицы, то в этом случае возникнет ошибка.

Возвращаемые значения содержат изменения, произведённые в триггерах BEFORE, но не в триггерах AFTER. OLD.fieldname и NEW.fieldname могут быть использованы в качестве имён столбцов. Если OLD. или NEW. не указано, возвращаются новые значения столбцов (NEW.). Вместо списка столбцов вы можете указать звёздочку (*). В этом случае будут возвращены все значения столбцов таблицы. Звёздочку можно применять со спецификаторами NEW или OLD.

```
UPDATE Scholars
SET first_name = 'Hugh' , last_name = 'Pickering'
WHERE first_name = 'Henry' AND last_name = 'Higgins'
RETURNING id, old.last_name, new.last_name;
```

Ключевое слово INTO позволяет сохранить эти возвращенные значения во внутренних переменных триггера или хранимой процедуры (только в PSQL). Если записи не было обновлены, ничего не возвращается, и переменные, указанные в RETURNING, сохранят свои прежние значения.

11.4 UPDATE OR INSERT

Оператор UPDATE OR INSERT позволяет изменить существующие данные или добавить новые, если в таблице нет строк, соответствующих некоторому условию.

Синтаксис оператора представлен в листинге 11.27.

Листинг 11.27. Синтаксис оператора изменения или добавления данных UPDATE OR INSERT

```
UPDATE OR INSERT INTO
   {<имя таблицы> | <имя представления>} [(<имя столбца> [, <имя столбца> ...] )]
VALUES (<значение> | DEFAULT [, <значение> | DEFAULT ...] )
[MATCHING (<имя столбца> [, <имя столбца>] ...)]
[RETURNING { <возвращаемые значения> | [{OLD. | NEW.}]* }
   [INTO [:]<имя переменной> [, [:]<имя переменной> ...]] ];

<возвращаемые значения> ::= <имя столбца> [[AS] <алиас>] [, <имя столбца> [[AS]
<алиас>]]...
```

Для выполнения оператора UPDATE OR INSERT пользователь должен иметь привилегии и UPDATE, и INSERT к таблице (представлению).

Совпадением считается полное совпадение значений столбцов MATCHING или PK. Совпадение проверяется с использованием IS NOT DISTINCT, поэтому NULL совпадает с NULL.

Синтаксис оператора немного похож на синтаксис оператора INSERT. После названия оператора и ключевого слова INTO помещается имя таблицы или представления, к которому применяется оператор. Затем в скобках следует необязательный список имен столбцов таблицы (представления). Ключевое слово VALUES является обязательным. После него в скобках следует список значений, присваиваемых соответствующим столбцам.

В списке VALUES вместо значения столбца можно использовать ключевое слово DEFAULT. В этом случае столбец получит значение по умолчанию, указанное при определении целевой таблицы. Если значение по умолчанию для столбца отсутствует, то столбец получит значение NULL.

Столбец для которого вместо значения использовано ключевое слово DEFAULT не может быть использован предложении MATCHING.

Оператор позволяет изменить значения отдельных столбцов в существующей строке или нескольких строках, если найдено соответствие, или добавить одну новую строку, если соответствия не найдено. В случае добавления новой строки в операторе должны быть заданы значения всех столбцов, входящих в состав первичного ключа, а также столбцов, описанных с характеристикой NOT NULL.

Если не задано ключевое слово MATCHING, то в списке столбцов обязательно должны присутствовать все столбцы, входящие в состав первичного ключа. Соответствующая строка будет найдена, если в таблице существует запись с тем же значением первичного ключа, что задано в операторе. В этом случае в строке произойдет изменение значений остальных указанных в операторе столбцов. Здесь изменяются только столбцы одной строки. Если строки с указанным в операторе значением первичного ключа не найдено, то в таблицу добавляется новая строка.

Если в операторе указано ключевое слово MATCHING и после него список имен столбцов, то поиск соответствующих строк осуществляется по этим столбцам (значения всех этих столбцов должны присутствовать в предложении VALUES). Если найдены соответствующие строки, то для них выполняется изменение значений указанных столбцов. Таких строк может быть более одной. Если не найдено ни одной соответствующей строки, то в таблицу добавляется одна новая строка. Если у таблицы нет первичного ключа, то указание MATCHING считается обязательным.

Предложение RETURNING позволяет вернуть значения указанных столбцов измененной или добавленной строки. Если изменяется или добавляется более одной строки, то наличие данного предложения вызовет исключение базы данных.

Возвращаемые значения содержат все изменения, произведённые в триггерах BEFORE, но не в триггерах AFTER. OLD.fieldname и NEW.fieldname могут быть использованы в качестве возвращаемых значений. Для обычных имён столбцов возвращаются новые значения.

Вместо списка столбцов вы можете указать звёздочку (*). В этом случае будут возвращены все значения столбцов таблицы. Звёздочку можно применять со спецификаторами NEW или OLD.

Ключевое слово INTO позволяет сохранить возвращенные значения во внутренних переменных триггера или хранимой процедуры. Здесь список также не заключается в скобки. Вариант INTO может использоваться только в PSQL.

11.5 DELETE

Для удаления существующих строк из таблиц базы данных используется оператор DELETE. Оператор позволяет удалить все или группу строк таблицы. Его синтаксис показан в листинге 11.28.

Листинг 11.28. Синтаксис оператора удаления данных из таблицы или представления DELETE

```
DELETE FROM {<имя таблицы> | <имя представления>} [[AS] <псевдоним>]
[WHERE { <условие поиска> | CURRENT OF <имя курсора>}]
[PLAN <план>]
[ORDER BY <упорядочиваемый элемент> [, <упорядочиваемый элемент> ... ]]
[ROWS <m> [TO <n>]]
[SKIP LOCKED]
[RETURNING {<список возвращаемых значений> | *} [INTO <переменные>]]

<список возвращаемых значений> ::= <имя столбца> [[AS] <алиас>] [, <имя столбца> [[AS]
<алиас>]...]

<переменные> ::= [:]<имя переменной> [, [:]<имя переменной> ...];
```

Удалять данные из таблицы (таблиц, лежащих в основе представления) может ее владелец, пользователь SYSDBA, пользователь операционной системы root (Linux), trusted user (Windows), а также пользователь, которому предоставлено право на удаление строк таблицы (таблиц) оператором GRANT DELETE — см. документ "Руководство администратора". Если удаление строки таблицы влечет и удаление подчиненных строк из дочерних таблиц, то и к этим таблицам пользователь также должен иметь соответствующие полномочия.

Оператор удаляет из таблицы строки в соответствии с условием в предложении WHERE и заданными значениями в предложениях ORDER BY и ROWS.

Предложение FROM задает имя таблицы или изменяемого представления, в которой удаляются строки.

11.5.1 WHERE

Предложение WHERE определяет множество строк, которые будут удалены. Удаляются только те строки, которые удовлетворяют условию поиска, или только текущей строке именованного курсора (только в PSQL). Если это предложение не указано, будут удалены все существующие строки таблицы в том случае, если не указано также предложение ORDER BY и предложение ROWS.

Для возможности использования предложения WHERE в операторе DELETE пользователь должен также иметь полномочия выборки данных из таблицы — GRANT SELECT.

Подробнее об условиях поиска в предложении WHERE см. в разделе 11.1.

```
DELETE FROM REGION
WHERE (CODCOUNTRY = 'RUS' AND CODREGION = 'TX');
```

11.5.2 PLAN

Может быть использовано ключевое слово PLAN, задающее план выборки данных для удаления. Подробнее о плане выборки см. в разделе 11.1.11. Основным назначение этого предложения является определение порядка (алгоритма) выборки строк из исходной таблицы для выполнения удаления. Такое предложение может ускорить (или замедлить при неправильном применении) выборку предназначенных для удаления записей.

11.5.3 ORDER BY и ROWS

Предложение `ORDER BY` используется для упорядочения результатов выборки перед его удалением. В нем указывается список столбцов, по которым происходит упорядочение, направление сортировки для каждого столбца (по возрастанию или по убыванию) и порядок сортировки (`COLLATE`) для строкового столбца. Синтаксис предложения `ORDER BY` и его использование подробно описано в разделе 11.3 этой главы.

```
MERGE INTO customers c
USING (select * from customers_delta where id > 10) cd
ON (c.id = cd.id)
WHEN MATCHED THEN
   UPDATE SET name = cd.name
WHEN NOT MATCHED THEN
   INSERT (id, name) VALUES (cd.id, cd.name);
```

Предложение `ROWS` позволяет ограничить количество удаляемых строк. Имеет смысл только в комбинации с предложением `ORDER BY`, но допустимо и без него.

В качестве `m` и `n` могут выступать любые целочисленные выражения.

При одном аргументе `m`, удаляются первые `m` строк. Порядок строк без `ORDER BY` не определён (случаен).

Замечания:

- Если `m` больше общего числа строк в наборе, то весь набор удаляется;
- Если `m = 0`, то удаление не происходит;
- Если `m < 0`, то выдаётся сообщение об ошибке.

Если указаны аргументы `m` и `n`, удаление ограничено количеством строк от `m` до `n`, включительно. Нумерация строк начинается с 1.

Замечания по использованию двух аргументов:

- Если `m > общего числа строк в наборе`, ни одна строка не удаляется;
- Если `m > 0` и `<= числа строк в наборе`, а `n` вне этих значений, то удаляются строки от `m` до конца набора;
- Если `m < 1` или `n < 1`, выдаётся сообщение об ошибке;
- Если `n = m - 1`, ни одна строка не удаляется;
- Если `n < m - 1`, выдаётся сообщение об ошибке.

11.5.4 RETURNING

Предложение `RETURNING` позволяет вернуть вызвавшей программе значения указанных столбцов удаленной строки. Если происходит удаление более чем одной строки, то выдается сообщение об ошибке.

В `RETURNING` могут быть указаны любые столбцы, не обязательно все, а также другие столбцы и выражения. Вместо списка столбцов может быть указана звёздочка (*), в этом случае будут возвращены все столбцы удалённой записи.

Ключевое слово `INTO` позволяет сохранить возвращенные значения во внутренних переменных триггера или хранимой процедуры. Вариант `INTO` может использоваться только в PSQL — см. главу 17.

11.6 MERGE

Объединяет данные в таблицу или представление.

Листинг 11.29. Синтаксис оператора MERGE

```
MERGE INTO <имя таблицы | имя представления> [[AS] <алиас>]
USING <таблица|представление|хранимая процедура|производная таблица> [AS <алиас>]
ON <условие соединения>
<предложение WHEN> [<предложение WHEN> ...]
[PLAN <выражение для построения плана>]
     [ORDER BY <выражение для упорядочивания выборки>]
[RETURNING <список возвращаемых выражений>|[{OLD. | NEW.}]*
   [INTO <список переменных>]]

<предложение WHEN> ::= <предложение WHEN MATCHED>
                         | <предложение WHEN NOT MATCHED BY TARGET>
                         | <предложение WHEN NOT MATCHED BY SOURCE>

<предложение WHEN MATCHED> ::=
       WHEN MATCHED [ AND <доп.условие> ]
       THEN {UPDATE SET <список назначений> | DELETE}

<предложение WHEN NOT MATCHED BY TARGET> ::=
       WHEN NOT MATCHED [ BY TARGET ] [ AND <доп.условие> ]
       THEN INSERT [(<столбцы>)] VALUES (<значения>)

<предложение WHEN NOT MATCHED BY SOURCE> ::=
       WHEN NOT MATCHED BY SOURCE [ AND <доп.условие> ]
       THEN { UPDATE SET <список назначений> | DELETE }

<список назначений> ::= <имя столбца>={<значение>|DEFAULT} [, <имя столбца>= {
<значение>|DEFAULT} ...]

<столбцы> ::= <имя столбца> [, <имя столбца> ...]

<значения> ::= { <значение> | DEFAULT} [, { <значение> | DEFAULT} ...]

<список возвращаемых выражений> ::= <выражение> [[AS] <псевдоним>] [, <выражение>
[[AS] <псевдоним>] ...]

<список переменных> ::= [:] <имя переменной> [, [:] <имя переменной> ...]
```

Оператор MERGE производит слияние записей источника в целевую таблицу (или обновляемое представление). Источником данных может быть таблица, представление, хранимая процедура или производная таблица, т.е. заключенный в скобки оператор SELECT. Каждая запись источника используется для обновления (предложение UPDATE) или удаления (предложение DELETE) одной или более записей цели, или вставки (предложение INSERT) записи в целевую таблицу, или ни для того, ни для другого. Условие обычно содержит сравнение столбцов в таблицах источника и цели.

В списке VALUES предложения INSERT и списке SET предложения UPDATE вместо значения столбца можно использовать ключевое слово DEFAULT. В этом случае столбец получит значение по умолчанию, указанное при определении целевой таблицы. Если значение по умолчанию для столбца отсутствует, то столбец получит значение NULL.

Допускается указывать несколько предложений `WHEN MATCHED` и `WHEN NOT MATCHED`.

Предложение `WHEN NOT MATCHED BY TARGET` вызывается, когда исходная запись не совпадает с ни с одной записью в целевой таблице. `INSERT` изменит целевую таблицу.

Предложение `WHEN NOT MATCHED BY SOURCE` вызывается, когда целевая запись не совпадает ни с одной записью в источнике. `UPDATE` или `DELETE` изменяют целевую таблицу.

Предложения `WHEN` проверяются в указанном порядке. Если условие в предложении `WHEN` не выполняется, то мы пропускаем его и переходим к следующему предложению. Так будет происходить до тех пор, пока условие для одного из предложений `WHEN` не будет выполнено. В этом случае выполняется действие, связанное с предложением `WHEN`, и осуществляется переход на следующую строку источника. Для каждой строки источника выполняется только одно действие.

Предложение `WHEN NOT MATCHED` берёт за основу записи из источника данных, указанного в предложении `USING`. Это значит, что если у исходной записи нет соответствия в целевой таблице, то выполняется оператор `INSERT`. С другой стороны записи в целевой таблице, не имеющие соответствия в источнике данных, не вызывают никаких действий.

Если условие `WHEN MATCHED` присутствует, и несколько записей совпадают с записями в целевой таблице, `UPDATE` выполнится для всех совпадающих записей источника, и каждое последующее обновление перезапишет предыдущее. Это нестандартное поведение: стандарт SQL-2003 требует, чтобы в этой ситуации выдавалось исключение (ошибка).

Оператор `MERGE`, затрагивающий не более одной строки, может содержать конструкцию `RETURNING` для возвращения значений добавленной, модифицируемой или удаляемой строки. В `RETURNING` могут быть указаны любые столбцы из целевой таблицы (обновляемого представления), не обязательно все, а также другие столбцы и выражения.

Возвращаемые значения содержат изменения, произведённые в триггерах `BEFORE`.

Имена столбцов могут быть уточнены с помощью префиксов `NEW` и `OLD` для определения, какое именно значение вы хотите столбца вы хотите получить до модификации или после.

Для предложений `WHEN MATCHED UPDATE` и `MERGE WHEN NOT MATCHED` неуточненные имена столбцов или уточнённые именами таблиц или их псевдонимами понимаются как столбцы с префиксом `NEW`, для предложений `MERGE WHEN MATCHED DELETE` – с префиксом `OLD`.

Пример:

```
MERGE INTO customers c
USING (select * from customers_delta where id > 10) cd
ON (c.id = cd.id)
WHEN MATCHED THEN
   UPDATE SET name = cd.name
WHEN NOT MATCHED THEN
   INSERT (id, name) VALUES (cd.id, cd.name);
```

Если присутствует предложение `WHEN MATCHED` и нескольким записям из источника данных соответствуют записи в целевой таблице, то операция обновления выполняется многократно; при этом каждое обновление перезаписывает предыдущее. Это поведение не соответствует стандарту: SQL-2003 определяет, что в таком случае должно быть вызвано исключение .

11.7 EXECUTE PROCEDURE

В DSQL, в языке хранимых процедур и триггеров и при использовании утилиты isql можно вызвать выполняемую хранимую процедуру, используя оператор EXECUTE PROCEDURE. Его синтаксис:

Листинг 11.30. Синтаксис оператора EXECUTE PROCEDURE

```
EXECUTE PROCEDURE <имя процедуры> [(<параметр> [, <параметр>] ...)]
[RETURNING_VALUES [:]<параметр> [, [:]<параметр>] ...];
```

При вызове хранимой процедуры можно после имени процедуры указать список входных параметров для процедуры. Если процедура получает параметры, то список входных параметров в операторе EXECUTE PROCEDURE является обязательным. При этом требуется полное соответствие количества параметров и их типов данных.

Оператор EXECUTE PROCEDURE является наиболее часто используемым стилем вызова хранимой процедуры, которая написана для модификации некоторых данных, код которой не содержит оператора SUSPEND. Такие хранимые процедуры могут возвратить набор данных, состоящий не более чем из одной строки. Этот набор может быть передан в переменные другой (вызывающей) процедуры с помощью предложения RETURNING_VALUES. Клиентские интерфейсы, как правило, имеют обертку API, которые могут извлекать выходные значения в однострочный буфер при вызове процедуры через EXECUTE PROCEDURE в DSQL.

При вызове с помощью EXECUTE PROCEDURE процедур другого типа (селективных процедур) будет возвращена только первая запись из результирующего набора, несмотря на то, что эта процедура скорее всего должна возвращать многострочный результат. "Селективные" хранимые процедуры должны вызываться с помощью оператора SELECT, в этом случае они ведут себя как виртуальные таблицы.

Если этот оператор вызывается из isql, то нельзя использовать предложение RETURNING_VALUES.

```
EXECUTE PROCEDURE MakeFullName
   ('Mr./Mrs.' || First_Name, Middle_Name, upper(Last_Name))
RETURNING_VALUES FullName;
```

11.8 EXECUTE BLOCK

Для выполнения из декларативного SQL (DSQL) некоторых императивных действий используются анонимные (безымянные) PSQL блоки. Заголовок анонимного PSQL блока опциально может содержать входные и выходные параметры. Тело анонимного PSQL блока может содержать объявление локальных переменных, курсоров и блок PSQL операторов. Анонимный PSQL блок не определяется и сохраняется как объект метаданных, в отличии от хранимых процедур и триггеров. Он не может обращаться сам к себе.

Как и хранимые процедуры анонимные PSQL блоки могут использоваться для обработки данных или для осуществления выборки из базы данных.

Синтаксис оператора представлен в листинге 11.31.

Листинг 11.31. Синтаксис оператора EXECUTE BLOCK

```
EXECUTE BLOCK
   [(<список входных параметров>)]
   [RETURNS (<список выходных параметров>)]
AS
   [<объявление> [<объявление> ...] ]
BEGIN
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
   <блок операторов>
END;

<список входных параметров>::=<описание параметра>=? [,<описание параметра>=?...]

<список выходных параметров> ::= <описание параметра> [, <описание параметра>]

<описание параметра> ::= <имя параметра> <тип> [NOT NULL]
                         [COLLATE <порядок сортировки>]

<тип> ::=  <тип данных SQL>
         | [TYPE OF] <имя домена>
         | TYPE OF COLUMN <имя таблицы/представления>.<имя столбца>

<объявление> ::=  объявление локальной переменной
                | объявление курсора
                | <объявление подпрограммы (процедуры или функции)>
                | <реализация подпрограммы (процедуры или функции)>
```

Оператор выполняет блок PSQL кода, так как будто это хранимая процедура, возможно с входными и выходными параметрами и локальными переменными. Это позволяет пользователю выполнять "на лету" PSQL в контексте DSQL.

Выполнение блока без входных параметров должно быть возможным с любым клиентом РЕД Базы Данных, который позволяет пользователю вводить свои собственные DSQL операторы. Если есть входные параметры, все становится сложнее: эти параметры должны получить свои значения после подготовки оператора, но перед его выполнением. Это требует специальных возможностей, которыми располагает не каждое клиентское приложение (`isql` такой возможности не предлагает).

Сервер принимает только вопросительные знаки (`"?"`) в качестве заполнителей для входных значений, а не `":xxx"` или литеральные значения. Клиентское программное обеспечение может поддерживать форму `":xxx"`, в этом случае будет произведена предварительная обработка запроса перед отправкой его на сервер.

Пример.

Вычисляет среднее геометрическое двух чисел и возвращает его пользователю:

```
EXECUTE BLOCK (
   x DOUBLE PRECISION = ?,
   y DOUBLE PRECISION = ?)
RETURNS (gmean DOUBLE PRECISION)
AS
BEGIN
   gmean = sqrt(x*y);
   SUSPEND;
END
```

Поскольку этот блок имеет входные параметры, он должен быть предварительно подготовлен. После чего можно установить параметры и выполнить блок.

Если блок имеет выходные параметры, нужно использовать `SUSPEND`, иначе ничего не будет возвращено. Выходные данные всегда возвращаются в виде набора данных, так же как и в случае с оператором `SELECT`. Не получится использовать `RETURNING_VALUES` или выполнить блок, вернув значения в некоторые переменные, используя `INTO`, даже если возвращается всего одна строка.

Типом данных для внутренней переменной (входного, выходного параметра или локальной пере-

менной) может быть любой тип данных, используемый в SQL.

Вместо типа данных можно указать имя домена. В этом случае внутренней переменной присваиваются все характеристики домена — запрет на пустое значение (NOT NULL), значение по умолчанию (DEFAULT) и условие (CHECK), которому должно удовлетворять значение, помещаемое в переменную. В случае задания в операторе предложения TYPE OF для этой переменной создается лишь тип данных, заданный в домене.

Внутренние можно объявлять, используя тип данных столбцов существующих таблиц и представлений. Для этого используется предложение TYPE OF COLUMN, после которого указывается имя таблиц или представления и через точку имя столбца. При использовании TYPE OF COLUMN наследуется только тип данных, а в случае строковых типов ещё набор символов и порядок сортировки. Ограничения и значения по умолчанию столбца никогда не используются.

Для внутренних переменных можно указать ограничение NOT NULL, тем самым запретив передавать в него значение NULL.

Для строковых типов данных, заданных явно или при ссылке на домен, можно указывать предложение COLLATE, определяющее порядок сортировки.

Локальной переменной можно устанавливать инициализирующее (начальное) значение. Это значение устанавливается с помощью предложения DEFAULT или оператора "=". В качестве значения по умолчанию может быть использовано значение NULL, литерал и любая контекстная переменная совместимая по типу данных.

Некоторые клиенты, особенно те, что позволяет пользователю отослать несколько операторов сразу, могут потребовать, окружить оператор EXECUTE BLOCK строками SET TERM.

Пример.

В следующем примере выполняется вычисление факториала числа, заданного входным параметром.

```
EXECUTE BLOCK
   (N BIGINT = ?)
RETURNS (RESULT BIGINT)
AS
   DECLARE VARIABLE I BIGINT;
BEGIN
   RESULT = 1;
   I = 1;
   WHILE (I <= N) DO
     BEGIN
        RESULT = RESULT * I;
        I = I + 1;
     END
   SUSPEND;
END
```

После выполнения цикла по вычислению факториала заданного числа выдается оператор SUSPEND, который позволяет отобразить полученный результат в программе графического интерфейса.

Подробное описание параметров, локальных переменных и операторов языка хранимых процедур и триггеров см. в главе 17. Там же приведен пример хранимой процедуры, выполняющей вычисление факториала заданного числа.

11.9 SET OPTIMIZE

Оператор определяет, что нужно оптимизировать: получение первых строк или всех.

Листинг 11.32. Синтаксис предложения SET OPTIMIZE

```
SET OPTIMIZE <режим оптимизации>

<режим оптимизации> ::= FOR {FIRST | ALL} ROWS
                      | TO DEFAULT
```

Оператор SET OPTIMIZE позволяет изменить стратегию оптимизатора на уровне текущей сессии.

Существует две стратегии оптимизации запросов:

- FIRST ROWS — оптимизатор строит план запроса так, чтобы быстрее извлечь только первые строки запроса;
- ALL ROWS — оптимизатор строит план запроса так, чтобы быстрее извлечь все строки запроса

По умолчанию используется стратегия оптимизации указанная в параметре OptimizeForFirstRows конфигурационного файла firebird.conf или database.conf.

Стратегия оптимизации может быть переопределена на уровне оператора с помощью предложения OPTIMIZE FOR.

Глава 12

Генераторы (GENERATOR/SEQUENCE)

Генератор (generator) или последовательность (sequence) — это самый простой объект базы данных. Он позволяет хранить целые числа в очень широком диапазоне значений: от -2^{63} до $2^{63}-1$.

Под него отводится 8 байтов памяти. Это подходящее средство для формирования значений искусственных первичных ключей. Для каждого искусственного первичного ключа любой таблицы базы данных пользователем создается свой собственный генератор, с которым выполняются все действия по формированию значений этого первичного ключа.

Важной особенностью генераторов является то, что работа с ними выполняется вне контекста какой-либо транзакции. Это означает, что при одновременном обращении к одному и тому же генератору разных конкурирующих транзакций никогда не возникнет конфликта блокировки, и каждый параллельный процесс получит уникальное новое числовое значение. Значение зависит от времени обращения к генератору.

В принципе, генераторы могут использоваться и для получения последовательностей неповторяющихся целых чисел для любых других целей.

12.1 Создание генератора

Для создания генератора используется оператор SQL CREATE GENERATOR/SEQUENCE. Синтаксис оператора показан в листинге 12.1.

Листинг 12.1. Синтаксис оператора создания генератора CREATE GENERATOR/ SEQUENCE

```
CREATE {GENERATOR | SEQUENCE} <имя генератора>
   [START WITH <начальное значение>] [INCREMENT [BY] <приращение>];
```

Ключевые слова GENERATOR и SEQUENCE являются синонимами (но рекомендуется использовать SEQUENCE).

Имя генератора должно быть уникальным среди имен всех генераторов базы данных и должно содержать до 63 символов.

В момент создания последовательности ей устанавливается текущее значение, указанное в необязательном предложении START WITH минус инкремент. Если предложение START WITH отсутствует, то последовательности устанавливается начальное значение равное 1.

> До версии 5.0 последовательности создавались с текущим значением равным стартовому значению (или 0 по умолчанию).
>
> Следующим значением последовательности со стартовым значением 0 и приращением 1 было 1.
>
> В версии 5.0 последовательности создаются (перезапускаются) с текущим значением равным стартовому минус инкремент. И начальным значением по умолчанию является 1 (а не 0).
>
> То есть результат оператора NEXT VALUES FOR для последовательности со стартовым значением 100 и приращением 10 будет 100 (а не 110, как было раньше). Аналогично функция GEN_ID(SEQ, 1) возвратит 100 (а не 101, как было раньше).

Необязательное предложение INCREMENT [BY] позволяет задать шаг приращения (4 байтное целое число) для оператора NEXT VALUES FOR. По умолчанию шаг приращения равен единице. Приращение не может быть установлено в ноль для пользовательских последовательностей. Значение последовательности изменяется также при обращении к функции GEN_ID, где в качестве параметра указывается имя последовательности и значение приращения, которое может быть отлично от указанного в пред-

ложении INCREMENT BY.

Создавать последовательности могут администраторы и те пользователи, у кого есть привилегия CREATE SEQUENCE (CREATE GENERATOR).

Пользователь, создавший последовательность, становится её владельцем.

```
CREATE SEQUENCE EMP_NO_GEN START WITH 5 INCREMENT BY 10;
```

12.2 Изменение значения генератора

Можно явно в любой момент времени установить новое значение генератора, выполнив оператор SET GENERATOR (листинг 12.2).

Листинг 12.2. Синтаксис оператора изменения значения генератора SET GENERATOR

```
SET GENERATOR <имя генератора> TO <значение>;
```

Для этой конструкции существует семантически одинаковая конструкция, которая выполняет те же действия — ALTER SEQUENCE (см. листинг 12.3). Именно этот вариант рекомендуется использовать в настоящей версии Ред База Данных.

Листинг 12.3. Альтернативный синтаксис оператора изменения значения генератора ALTER SEQUENCE

```
ALTER SEQUENCE <имя генератора>
   [START WITH <значение>]
   [RESTART [WITH <значение>]]
   [INCREMENT [BY] <приращение>];
```

Предложение START WITH изменяет начальное значение последовательности.

Предложение RESTART WITH позволяет установить значение последовательности (см. замечание из подраздела 12.1).

Предложение RESTART может быть использовано самостоятельно (без WITH) для перезапуска значения последовательности с начального значения или со значения, с которого начиналась генерация при предыдущем рестарте.

Предложение INCREMENT [BY] позволяет изменить шаг приращения последовательности для оператора NEXT VALUES FOR.

Изменение значения приращения — это возможность, которая вступает в силу для каждого запроса, который запускается после фиксаций изменения. Процедуры, которые вызваны впервые после изменения приращения, будут использовать новое значение, если они будут содержать операторы NEXT VALUE FOR. Процедуры, которые уже работают, не будут затронуты, потому что они кэшируются. Процедуры, использующие NEXT VALUE FOR, не должны быть перекомпилированы, чтобы видеть новое приращение, но если они уже работают или загружены, то никакого эффекта не будет. Конечно процедуры, использующие GEN_ID, не затронут при изменении приращения.

Нет особой необходимости использовать описанные операторы. Более того, разработчиками системы не рекомендуется вообще использовать эти операторы, поскольку это может привести к нарушениям в базе данных при помещении в разные строки первичного ключа таблицы одинаковых значений. Тем не менее, такие операторы были введены в SQL для большего соответствия стандарту SQL-99. Более естественной и безопасной конструкцией является конструкция NEXT VALUE FOR без каких-либо неправильных модификаций значения генератора.

Когда значение генератора достигает максимальной величины, то все новые обращения к нему переводят его значение в отрицательную величину. В этот момент при каждом новом обращении к генератору начинается отрицательный отсчет от максимальной его величины (–9,223,372,036,854,775,808) к нулю.

Операторы `ALTER SEQUENCE (GENERATOR)` и `SET GENERATOR` могут выполнять владельцы последовательностей, администраторы и пользователи с привилегией `ALTER ANY SEQUENCE (GENERATOR)`.

12.3 Создание нового или изменение существующего генератора

С помощью оператора `CREATE OR ALTER GENERATOR (SEQUENCE)` можно создать новую или изменить существующую последовательность:

Листинг 12.4. Синтаксис оператора `CREATE OR ALTER GENERATOR /SEQUENCE`

```
CREATE OR ALTER {GENERATOR | SEQUENCE} <имя генератора>
   [{START WITH <начальное значение> | RESTART}]
   [INCREMENT [BY] <приращение>];
```

Если последовательности не существует, то она будет создана. Уже существующая последовательность будет изменена, при этом существующие зависимости последовательности будут сохранены.

Оператор `CREATE OR ALTER SEQUENCE` требует, чтобы хотя бы одно из необязательных предложений было указано.

12.4 Удаление генератора

Генератор можно удалить, используя оператор SQL `DROP GENERATOR / SEQUENCE`. Его синтаксис представлен в листинге 12.5.

Листинг 12.5. Синтаксис оператора удаления генератора `DROP GENERATOR/SEQUENCE`

```
DROP {GENERATOR | SEQUENCE} <имя генератора>;
```

При наличии зависимостей для существующей последовательности (генератора) удаления не будет выполнено. Удалять генератор следует только после того, как будут удалены из базы данных все триггеры и хранимые процедуры, ссылающиеся на этот генератор.

Удалять генераторы могут администраторы, владельцы последовательности и пользователи с привилегией `DROP ANY SEQUENCE` (`GENERATOR`).

Практическое использование генераторов для формирования значений первичного ключа в таблицах см. в главе 11.

12.5 Пересоздание генератора

Последовательность можно пересоздать с помощью оператора RECREATE GENERATOR (SEQUENCE):

Листинг 12.6. Синтаксис оператора пересоздания генератора RECREATE GENERATOR /SEQUENCE

```
RECREATE {GENERATOR | SEQUENCE} <имя генератора>
   [START WITH <начальное значение>] [INCREMENT [BY] <приращение>];
```

Если последовательность с таким именем уже существует, то оператор RECREATE SEQUENCE попытается удалить её и создать новую последовательность. При наличии зависимостей для существующей последовательности оператор RECREATE SEQUENCE не выполнится.

12.6 Примечание к генератору

Для генератора также можно создать примечание, используя следующий синтаксис оператора COMMENT (см. листинг 12.7).

Листинг 12.7. Синтаксис оператора создания примечания генератора

```
COMMENT ON
   {GENERATOR | SEQUENCE} <имя генератора> IS {'<текст>' | NULL};
```

Чтобы создать примечание для генератора GEN_PEOPLE, нужно выполнить следующий оператор:

```
COMMENT ON
    GENERATOR GEN_PEOPLE IS 'Используется для генерации первичного ключа в PEOPLE';
```

Глава 13

BLOB фильтр (FILTER)

BLOB фильтр — объект базы данных, являющийся, по сути, специальным видом внешних функций с единственным назначением: получение объекта BLOB одного формата и преобразования его в объект BLOB другого формата. Форматы объектов BLOB задаются с помощью подтипов BLOB.

Внешние функции для преобразования BLOB типов хранятся в динамических библиотеках и загружаются по необходимости.

13.1 Объявление BLOB фильтра

Оператор DECLARE FILTER объявляет существующий BLOB фильтр в базе данных.

Листинг 13.1. Синтаксис оператора объявления BLOB фильтра

```
DECLARE FILTER <имя фильтра>
INPUT_TYPE <подтип BLOB> OUTPUT_TYPE <подтип BLOB>
ENTRY_POINT 'Имя экспортируемой функции'
MODULE_NAME 'Имя модуля с фильтром'

<подтип BLOB> ::= номер подтипа | <мнемоника подтипа>

<мнемоника подтипа> ::= binary | text | blr | acl | ranges | summary |
                        format | transaction_description |
                        external_file_description | <user_defined>
```

Имя BLOB фильтра должно быть уникальным среди имен BLOB фильтров и может содержать до 63 символов.

Создать BLOB фильтр может администратор и пользователь с привилегией CREATE FILTER.

Пользователь, создавший BLOB фильтр, становится его владельцем.

13.1.1 Задание подтипов

Предложение INPUT_TYPE устанавливает подтип BLOB преобразуемого объекта. Предложение OUTPUT_TYPE устанавливает подтип создаваемого объекта. Подтип задается в виде номера подтипа или мнемоники подтипа. Пользовательские подтипы должны быть представлены отрицательными числами (от -1 до -32768).

В базе данных не может быть двух и более фильтров BLOB с одинаковыми комбинациями входных и выходных типов. Объявление фильтра с уже существующими комбинациями входных и выходных типов BLOB приведет к ошибке.

Для определения мнемоники для собственных подтипов BLOB, можно добавить их в системную таблицу RDB$TYPES:

```
INSERT INTO RDB$TYPES (RDB$FIELD_NAME, RDB$TYPE, RDB$TYPE_NAME)
VALUES ('RDB$FIELD_SUB_TYPE', -33, 'MIDI');
```

После подтверждения транзакции мнемоники могут быть использованы для декларации при создании новых фильтров.

Значение поля RDB$FIELD_NAME всегда должно быть подтипа RDB$FIELD_SUB_TYPE. При определении мнемоники в верхнем регистре можно использовать их без учета регистра и без кавычек при объявлении фильтра.

13.1.2 Другие параметры

Предложение ENTRTY_POINT указывает имя экспортируемой функции (точки входа) в модуле.

Предложение MODULE_NAME задает имя модуля, в котором находится экспортируемая функция. По умолчанию модули должны располагаться в папке UDF корневого каталога сервера. Параметр UDFAccess в файле firebird.conf позволяет изменить ограничения доступа к библиотекам фильтрам.

Пример 1.

Создание BLOB фильтра с использованием номеров подтипов.

```
DECLARE FILTER DESC_FILTER
INPUT_TYPE 1
OUTPUT_TYPE -4
ENTRY_POINT 'desc_filter'
MODULE_NAME 'FILTERLIB';
```

Пример 2.

Создание BLOB фильтра с использованием мнемоник подтипов.

```
DECLARE FILTER FUNNEL
INPUT_TYPE blr
OUTPUT_TYPE text
ENTRY_POINT 'blr2asc'
MODULE_NAME 'myfilterlib';
```

13.2 Удаление объявления BLOB фильтра

Данный оператор удаляет объявление BLOB фильтра из базы данных.

Листинг 13.2. Синтаксис оператора DROP FILTER

```
DROP FILTER <имя фильтра>
```

Удаление BLOB фильтра из базы данных делает его недоступным из базы данных. Но динамическая библиотека, в которой расположена функция преобразования, остается нетронутой.

Удалить объявление BLOB фильтра администратор, владелец фильтра и пользователь с привилегией DROP ANY FILTER.

Глава 14

Индексы (INDEX)

Индекс — это объект базы данных, содержащий значения указанных столбцов конкретной таблицы и ссылки на строки этой таблицы, содержащие данные значения. Индекс создается пользователем или системой для конкретной таблицы, что позволяет во многих случаях ускорить процесс поиска данных в этой таблице, а иногда и ускорить упорядочение данных, полученных по запросу пользователя на основании предложения ORDER BY в операторе SELECT. Каждая строка индекса содержит значение столбцов, входящих в состав индекса и указатель на строку в таблице, которая имеет те же самые значения столбцов. Кроме того, индекс может быть использован при определении ограничений, таких как первичный ключ, внешний ключ или ограничениях уникальности.

При наличии индексов во многих случаях поиск данных может выполняться гораздо быстрее, чем при отсутствии индекса, потому что значения в индексе упорядочены, а сам индекс относительно мал. Не следует создавать индексы для столбцов, которые имеют небольшое количество вариантов значений, например для столбцов, имеющих два значения, в частности, для столбцов, моделирующих логический тип данных, где столбец может иметь только значения TRUE и FALSE, или в случае задания пола человека — мужской или женский. Такие индексы только занимают место во внешней памяти и не дают никакого выигрыша в производительности при выполнении операций выборки и упорядочения данных.

Для ограничений первичного ключа, уникального ключа и внешнего ключа система автоматически строит индексы.

Как правило, использование индексов не является столь важной задачей в такой системе управления базами данных, как РЕД База Данных, поскольку сервер базы данных имеет возможности оптимизации своей работы и при отсутствии соответствующих индексов.

> Нельзя создавать индекс по структуре и по упорядоченности соответствующий индексу, который автоматически создается системой для первичного, уникального или внешнего ключа, при попытке выборки данных это может привести к аварийному завершению работы сервера базы данных.

Индекс может быть создан как уникальный (ключевое слово UNIQUE). В этом случае в таблице не допускается присутствие двух различных строк, имеющих одинаковое значение столбцов, входящих в состав уникального индекса.

Может быть создан частичный индекс, который действует только для записей, удовлетворяющих заданному условию.

Индекс может быть упорядочен по возрастанию значений столбцов, входящих в его состав (ASCENDING — значение по умолчанию) или по убыванию этих значений (DESCENDING). В любой момент времени работы с базой данных индекс может быть сделан активным (ACTIVE), то есть все изменения столбцов таблицы, входящих в состав этого индекса, тут же отражаются в самом индексе, или неактивным (INACTIVE), когда никакие изменения в строках соответствующей таблицы базы данных не затрагивают содержание индекса.

14.1 Создание индекса

Для создания индекса для существующей таблицы базы данных используется оператор CREATE INDEX. Его синтаксис представлен в листинге 14.1.

Листинг 14.1. Синтаксис оператора создания индекса CREATE INDEX

```
CREATE [UNIQUE] [ASC[ENDING] | DESC[ENDING]]
INDEX <имя индекса> ON <таблица>
{(<столбец> [, <столбец> ...]) | COMPUTED BY (<выражение>)}
[WHERE <условие>]
[[IN] TABLESPACE {<имя табличного пространства> | PRIMARY}];
```

Создать индекс может только владелец таблицы, для которой создан индекс, администратор и пользователь с привилегией ALTER ANY TABLE.

В состав индекса не могут входить вычисляемые поля, а также столбцы, имеющие тип данных BLOB и столбцы любого типа данных, являющиеся массивами.

Имя индекса должно быть уникальным среди имен всех индексов базы данных, а также среди имен ограничений на уровне столбцов таблицы и ограничений на уровне таблиц. Когда при задании ограничений первичного, уникального или внешнего ключа (см. главу 9) вы указываете и имя ограничения в предложении CONSTRAINT, система строит индекс с тем же самым именем. Если же при описании этих ключей задается и предложение USING, то автоматически создаваемый индекс получает имя, указанное в предложении USING.

Ключевое слово UNIQUE задает создание уникального индекса, оно указывает, что в индексе не может быть двух строк с одинаковыми значениями всех столбцов индекса. Но уникальные индексы могут содержать дубликаты значения NULL в соответствии со стандартом SQL-99 (в том числе и в многосегментном индексе).

Ключевое слово ASCENDING (сокращенный вариант ASC) означает, что записи индекса упорядочиваются по возрастанию значений столбцов, входящих в состав индекса. Этот вариант принимается по умолчанию.

Ключевое слово DESCENDING (сокращение DESC) указывает, что записи индекса упорядочиваются по уменьшению значений столбцов индекса.

Пример.

Если для таблицы PEOPLE требуются и возрастающий и убывающий индексы по столбцу, хранящему фамилии людей LAST_NAME, то нужно создать два индекса, выполнив операторы:

```
CREATE ASCENDING INDEX ASC_PEOPLE ON PEOPLE (LAST_NAME);
CREATE DESCENDING INDEX DESC_PEOPLE ON PEOPLE (LAST_NAME);
```

При создании индекса вместо одного или нескольких столбцов также можно указать одно выражение, используя предложение COMPUTED BY. Такой индекс называется вычисляемым или индексом по выражению. Вычисляемые индексы используются в запросах, в которых условие в предложениях WHERE, ORDER BY или GROUP BY в точности совпадает с выражением в определении индекса. Выражение в вычисляемом индексе может использовать несколько столбцов таблиц.

```
CREATE INDEX IDX_NAME_UPPER ON PERSONS
COMPUTED BY (UPPER (NAME));
```

Предложение WHERE позволяет создать индекс, распространяющийся только на записи таблицы, удовлетворяющие указанному условию. Частичный индекс может быть создан как уникальный (ключевое слово UNIQUE). В этом случае, каждый ключ в индексе должен быть уникальным. Это позволяет обеспечить уникальность для некоторого подмножества строк таблицы. Частичный индекс будет использован только в следующих случаях:

- Если условие WHERE содержит точно такое же выражение, как и то, которое определено для индекса;
- Если условие, определенное для индекса, содержит несколько выражений, объединенных опе-

ратором `OR`, и одно из них явно включено в условие `WHERE`;

- Если условие, определенное для индекса, содержит `IS NOT NULL`, а условие `WHERE` включает выражение для того же поля, про которое известно, что оно игнорирует `NULL`.

Пример.

```
CREATE INDEX IT1_COL ON T1 (COL) WHERE COL < 100;
```

Для индекса может быть указано табличное пространство для отдельного физического хранения.

```
CREATE INDEX idx_name ON PERSONS (NAME) TABLESPACE tablespace_name;
```

По умолчанию все индексы таблицы создаются в том же табличном пространстве, что и сама таблица.

Операторы перемещения индекса в табличное пространство требуют наличия единственного подключения к базе данных. Это временное ограничение, что делает процедуру перемещения более надежной.

14.1.1 Ограничения на индексы

Максимальная длина ключа индекса составляет $1/4$ размера страницы. Максимальная длина индексируемой строки на 9 байтов меньше, чем максимальная длина ключа. Максимальная длина индексируемой строки зависит от размера страницы и набора символов:

Размер страницы	**Максимальная длина индексируемой строки для набора символов, байт/символ**				
	1	**2**	**3**	**4**	**6**
4096	1015	507	338	253	169
8192	2039	1019	679	509	339
16384	4087	2043	1362	1021	682
32768	9183	4087	2721	2039	1356

Для каждой таблицы максимально возможное количество индексов ограничено и зависит от размера страницы и количества столбцов в индексе:

Размер страницы	**Число индексов в зависимости от количества столбцов в индексе**		
	1	**2**	**3**
4096	203	145	113
8192	408	291	227
16384	818	584	454

14.2 Изменение индекса

При первоначальном создании индекс становится по умолчанию активным — все вновь добавленные строки в таблицу или выполненные изменения в индексированных столбцах базовой таблицы тут же отражаются на состоянии индекса.

В некоторых случаях бывает полезным на некоторое время «отключить» индекс, сделать его

неактивным. Это может сэкономить время при выполнении так называемых пакетных операций с таблицей, когда в таблицу, для которой создан индекс, из какого-либо файла записывается достаточно большое количество строк или в таблице изменяется или из таблицы удаляется большое количество строк. Перед началом такой операции индекс переводится в неактивное (`INACTIVE`) состояние, а после завершения операции — снова в активное (`ACTIVE`). При этом при активизации индекса осуществляется полное пересоздание индекса. Все вновь введенные, измененные или удаленные строки будут учтены в новом состоянии индекса.

Изменение состояния индекса осуществляется при помощи оператора `ALTER INDEX`. Его синтаксис представлен в листинге 14.2. Этот оператор не может быть использован для изменения структуры индекса или его упорядоченности. Если есть необходимость внести изменения в структуру индекса или изменить порядок, то следует удалить существующий индекс (см. следующий раздел), а затем создать индекс с тем же именем и с требуемыми характеристиками.

Листинг 14.2. Синтаксис оператора изменения индекса `ALTER INDEX`

```
ALTER INDEX <имя индекса>
{ { ACTIVE | INACTIVE }
| SET TABLESPACE [TO] {<имя табличного пространства> | PRIMARY}
}
```

Состояние индекса может изменять только владелец таблицы, для которой создан индекс, администратор и любой пользователь с привилегией `ALTER ANY TABLE`.

Ключевое слово `INACTIVE` указывает, что индекс переводится в неактивное состояние. Перевод индекса в неактивное состояние по своему действию похоже на команду `DROP INDEX` за исключением того, что определение индекса сохраняется в базе данных. Невозможно перевести в неактивное состояние индекс участвующий в ограничении.

Активный индекс может быть отключен, только если отсутствуют запросы использующие этот индекс, иначе будет возвращена ошибка `«object in use»`.

Ключевое слово `ACTIVE` задает перевод неактивного индекса в активное состояние. После перевода индекса из неактивного в активное состояние система заново полностью создает весь индекс. Активация неактивного индекса безопасна. Тем не менее, если есть активные транзакции, модифицирующие таблицу, то транзакция, содержащая оператор `ALTER INDEX` потерпит неудачу, если она имеет атрибут `NO WAIT`. Если транзакция находится в режиме `WAIT`, то она будет ждать завершения параллельных транзакций.

С другой стороны, если оператор `ALTER INDEX` начинает перестраивать индекс на `COMMIT`, то другие транзакции, изменяющие эту таблицу, потерпят неудачу или будут ожидать в соответствии с их `WAIT/NO WAIT` атрибутами. Та же самая ситуация будет и при выполнении `CREATE INDEX`.

Даже если индекс находится в активном состоянии оператор `ALTER INDEX ... ACTIVE` всё равно перестраивает индекс. Таким образом, эту команду можно использовать как часть обслуживания БД, для индексов большой таблицы в которую происходят частые вставки, обновления и удаления. Для перестройки индексов, автоматически созданных для ограничений `PRIMARY KEY, FOREIGN KEY, UNIQUE`, для которых выполнение оператора `ALTER INDEX ... INACTIVE` невозможно.

Принудительный перевод индексов, созданных для ограничений `PRIMARY KEY, FOREIGN KEY` и `UNIQUE` не допускается. Тем не менее, выполнение оператора `ALTER INDEX ... INACTIVE` работает так же хорошо для индексов ограничений как и другие инструменты для других индексов.

Для индекса может быть указано табличное пространство для отдельного физического хранения.

```
ALTER INDEX idx_name SET TABLESPACE TO tablespace_name;
```

По умолчанию все индексы таблицы создаются в том же табличном пространстве, что и сама таблица.

Чтобы переместить индекс из табличного пространства в основной файл базы данных, воспользуйтесь командой:

```
ALTER INDEX idx_name SET TABLESPACE TO PRIMARY;
```

Если индекс находился в неактивном состоянии, то после назначения ему другого `TABLESPACE` он останется неактивным. Только после активации страницы индекса будут созданы в новом табличном пространстве.

Операторы перемещения индекса в табличное пространство требуют наличия единственного подключения к базе данных. Это временное ограничение, что делает процедуру перемещения более надежной.

Пример.

Чтобы перевести индекс `DESC_PEOPLE` перед выполнением какой-либо пакетной операции в неактивное состояние, нужно выполнить следующий оператор:

```
ALTER INDEX DESC_PEOPLE INACTIVE;
```

После завершения соответствующих действий нужно перевести его в активное состояние, выполнив:

```
ALTER INDEX DESC_PEOPLE ACTIVE;
```

14.3 Удаление индекса

Для удаления индекса, созданного пользователем, используется оператор `DROP INDEX`. Его синтаксис представлен в листинге 14.3.

Листинг 14.3. Синтаксис оператора удаления индекса `DROP INDEX`

```
DROP INDEX <имя индекса>;
```

Нельзя таким образом удалить индекс, созданный автоматически системой для первичного, уникального или внешнего ключа. Можно только удалить индекс, который был создан пользователем.

При наличии зависимостей для существующего индекса (если он используется в ограничении) удаление не будет выполнено.

Удалить индекс может только владелец таблицы, для которой создан индекс, администратор и пользователь с привилегией `ALTER ANY TABLE`.

Пример.

Чтобы удалить индекс `DESC_PEOPLE`, нужно выполнить следующий оператор:

```
DROP INDEX DESC_PEOPLE;
```

14.4 Селективность индекса

Селективность (избирательность) индекса — это некоторое состояние, задаваемое числовым значением, которое определяет эффективность использования данного индекса при выборке данных. Селективность определяется числом от нуля до единицы. Чем меньше это число, тем выше селективность (полезность) индекса, тем выше эффективность использования индекса для поиска записей.

Селективность индекса — это оценочное количество строк, которые могут быть выбраны при поиске по каждому значению индекса. Уникальный индекс имеет максимальную селективность, поскольку при его использовании невозможно выбрать более одной строки для каждого значения ключа индекса. Актуальность селективности индекса важна для выбора наиболее оптимального плана выполнения запросов оптимизатором.

В процессе работы с базой данных, при добавлении новых строк, удалении существующих записей или при изменении значений столбцов, входящих в состав индекса, значение селективности может изменяться в худшую сторону. Улучшить селективность всех индексов можно, выполнив резервное копирование и последующее восстановление базы данных. См. документ «Руководство администратора». Улучшение селективности только одного конкретного индекса можно получить, выполнив оператор `SET STATISTICS` для этого индекса. Выполнение оператора приводит к тому, что индекс становится максимально селективным в конкретной таблице. Синтаксис оператора представлен в листинге 14.4.

Листинг 14.4. Синтаксис оператора изменения селективности индекса `SET STATISTICS`

```
SET STATISTICS INDEX <имя индекса>;
```

Оператор улучшает, оптимизирует селективность указанного индекса.

> Только владелец таблицы, для которой был создан индекс, администратор и пользователь с ролью `ALTER ANY TABLE` имеют привилегии на использование `SET STATISTICS INDEX`.

14.5 Примечание индекса

Для индекса можно создать примечание, используя следующий вариант оператора `COMMENT` (листинг 14.5).

Листинг 14.5. Синтаксис оператора создания примечания индекса `COMMENT ON INDEX`

```
COMMENT ON
   INDEX <имя индекса> IS {'<текст>' | NULL};
```

Если в качестве текста примечания задать `NULL`, то будет удалено существующее примечание индекса.

Пример.

Чтобы добавить примечание к индексу `ASC_PEOPLE`, нужно выполнить оператор:

```
COMMENT ON
    INDEX ASC_PEOPLE IS 'Индекс, упорядоченный по возрастанию фамилий людей';
```

Глава 15

Представления (VIEW)

Представление (view) — это объект базы данных, хранящийся в области метаданных. Другое название для представления, которое можно встретить в литературе: просмотр, обзор. Представление — виртуальная (реально не существующая) таблица, которая в базе данных не хранится. Основой представления является оператор SELECT произвольной сложности, который задает выборку данных из одной или более таблиц, других представлений, а также селективных хранимых процедур. В базе данных хранится оператор SELECT, но не результаты его выполнения. Результат в виде набора данных создается при обращении к представлению. К представлениям можно обращаться в операторах SELECT, представления могут принимать участие в операциях объединения (UNION) и соединения (JOIN), заданных в операторе выборки данных.

Представления, задавая выборку не всех, а только отдельных столбцов и строк исходных таблиц, дают возможность скрыть от рядового пользователя системы некоторые данные, не предназначенные для широкого просмотра. Например, это могут быть оклады сотрудников, пароли, некоторые количественные характеристики деятельности организации и др. В этом отношении представления являются хорошим средством повышения безопасности данных.

15.1 Создание представлений

Для создания представления используется оператор CREATE VIEW. Синтаксис оператора показан в листинге 15.1.

Листинг 15.1. Синтаксис оператора создания представления CREATE VIEW

```
CREATE VIEW <имя представления> [<список столбцов>]
AS <оператор SELECT>
[WITH CHECK OPTION]

<список столбцов> ::= ( <столбец> [,  <столбец> ...])
```

Представления могут создаваться администраторами и пользователями с привилегией CREATE VIEW.

Для создания представления пользователями, которые не имеют административных привилегий, необходимы также привилегии на чтение (SELECT) данных из базовых таблиц и представлений, и привилегии на выполнение (EXECUTE) используемых селективных хранимых процедур. Для разрешения вставки, обновления и удаления через представление, необходимо чтобы создатель (владелец) представления имел привилегии INSERT, UPDATE и DELETE на базовые объекты метаданных.

Пользователь, создавший представление, становится его владельцем.

Имя представления должно быть уникальным среди имен всех представлений, таблиц и хранимых процедур базы данных.

15.1.1 Поля представления

После имени создаваемого представления может идти список имен полей представления, заключенный в скобки. Если в представлении присутствуют элементы, значения которых получаются из выражений, то такой список столбцов является обязательным. В остальных случаях список можно не указывать. Обращение к столбцам можно осуществлять по именам столбцов, которые указаны в

списке выбора главного оператора SELECT в представлении. Однако хорошей практикой является явное задание списка столбцов в самом представлении. Имена в списке могут быть никак не связаны с именами столбцов базовых таблиц. При этом их количество должно точно соответствовать количеству столбцов в списке выбора главного оператора SELECT представления. По этим именам в списке к столбцам полученного набора данных можно обращаться в операторе SELECT, вызывающем данное представление.

15.1.2 Предложение AS

После ключевого слова AS следует оператор SELECT. Здесь можно выполнять объединение (UNION) и соединение (JOIN) различных таблиц, использовать предложение WHERE для задания условий выбора строк. Возможности оператора SELECT см. в главе 11.

15.1.3 Изменяемые представления

Представление может быть изменяемым (в двух вариантах — естественно изменяемым или изменяемым при помощи вспомогательных триггеров) или неизменяемым, только для чтения (read-only). В случае естественно изменяемого представления в данные, полученные при помощи такого представления (в базовую таблицу представления, то есть в таблицу, из которой представление получает все данные), пользователь может свободно вносить любые изменения, используя операторы INSERT, UPDATE, DELETE, UPDATE OR INSERT, MERGE. Выполненные изменения тут же помещаются в таблицу. При неизменяемом представлении пользователь не может вносить обычными средствами изменения в выбранные данные. Во многих случаях и в неизменяемое представление (в базовые таблицы) можно вносить изменения при использовании вспомогательных триггеров. Использование триггеров для получения изменяемых представлений из неизменяемых см. в разделе 15.6 данной главы.

Чтобы представление было естественно изменяемым, необходимо выполнение следующих условий:

- оператор SELECT выборки данных обращается только к одной таблице или к одному другому изменяемому представлению;
- оператор выборки SELECT не должен обращаться к хранимым процедурам;
- все столбцы исходной (базовой) таблицы или исходного изменяемого представления, которые не присутствуют в данном представлении, должны удовлетворять одному из следующих условий:
  - позволять значение NULL;
  - NOT NULL столбцы должны иметь значение по умолчанию;
  - значение NOT NULL столбцов должны быть инициализированы в триггерах базовых таблиц;
- оператор выборки SELECT не содержит полей определённых через подзапросы или другие выражения;
- оператор выборки SELECT не содержит полей определённых через агрегатные функции (MIN, MAX, AVG и др.), статистические функции (CORR, COVAR_POP, COVAR_SAMP и др.), функции линейной регрессии (REGR_AVGX, REGR_AVGY и др.) и все виды оконных функций;
- оператор выборки SELECT не содержит предложений ORDER BY, GROUP BY, HAVING;
- оператор выборки SELECT не содержит ключевого слова DISTINCT и ограничений количества строк ROWS, FIRST, SKIP, OFFSET, FETCH.

15.1.4 Предложение WITH CHECK OPTION

Необязательное предложение `WITH CHECK OPTION` задает для изменяемого представления требование проверки соответствия вновь вводимых или изменяемых данных условию, заданному в предложении `WHERE`. Если будет попытка поместить новую строку, которая не соответствует условию выборки в предложении `WHERE`, то такая строка не помещается в таблицу, выдается соответствующее диагностическое сообщение. Точно так же в этом случае недопустимы операции изменения полученных из представления данных, которые приводят к нарушению условия выборки в предложении `WHERE`.

Предложение `WITH CHECK OPTION` может задаваться в операторе создания представления только в том случае, если в операторе `SELECT` представления указано предложение `WHERE`. Иначе вы получите сообщение об ошибке.

Если используется предложение `WITH CHECK OPTIONS`, то система проверяет входные значение на соответствие условию в предложении `WHERE` до того как они будут переданы в базовую таблицу. Таким образом, если входные значения не проходят проверку, то предложения `DEFAULT` или триггеры на базовой таблице, не могут исправить входные значения, поскольку действия никогда не будут выполнены.

Кроме того, поля представления не указанные в операторе `INSERT` передаются в базовую таблицу как значения `NULL`, независимо от их наличия или отсутствия в предложении `WHERE`. В результате значения по умолчанию, определённые на таких полях базовой таблицы, не будут применены. С другой стороны, триггеры будут вызываться и работать как ожидалось.

Для представлений у которых отсутствует предложение `WITH CHECK OPTIONS`, поля, отсутствующие в операторе `INSERT`, не передаются вовсе, поэтому любые значения по умолчанию будут применены.

15.2 Изменение представлений

Для изменения существующего представления используется оператор `ALTER VIEW`. Синтаксис оператора показан в листинге 15.2.

Листинг 15.2. Синтаксис оператора создания представления `ALTER VIEW`

```
ALTER VIEW <имя представления> [<список столбцов>]
AS <оператор SELECT>
[WITH CHECK OPTION]

<список столбцов> ::= (столбец [, столбец ...])
```

Оператор `ALTER VIEW` изменяет определение существующего представления, существующие права на представления и зависимости при этом сохраняются. Синтаксис оператора `ALTER VIEW` полностью аналогичен синтаксису оператора `CREATE VIEW`.

Изменить представление могут только владелец представления, администратор, пользователь с привилегией `ALTER ANY VIEW`.

Будьте осторожны при изменении количества столбцов представления. Существующий код приложения может стать неработоспособным. Кроме того, PSQL модули, использующие изменённое представление, могут стать некорректными. Информация о том, как это обнаружить, находится в разделе Приложение B.

15.3 Создание или изменение представлений

Для создание нового или изменение существующего представления используется оператор `CREATE OR ALTER VIEW`. Синтаксис оператора показан в листинге 15.3.

Листинг 15.3. Синтаксис оператора создания представления `CREATE OR ALTER VIEW`

```
CREATE OR ALTER VIEW <имя представления> [<список столбцов>]
AS <оператор SELECT>
[WITH CHECK OPTION]

<список столбцов> ::= (столбец [, столбец ...])
```

Оператор `CREATE OR ALTER VIEW` создаёт представление, если оно не существует. В противном случае он изменит представление с сохранением существующих зависимостей.

15.4 Удаление представлений

Для удаления существующего в базе данных представления используется оператор `DROP VIEW`. Его синтаксис представлен в листинге 15.4.

Листинг 15.4. Синтаксис оператора удаления представления `DROP VIEW`

```
DROP VIEW <имя представления>;
```

Представление нельзя удалить, если на него есть ссылки в другом представлении, в хранимой процедуре или в ограничении `CHECK` столбца таблицы или соответствующего ограничения на уровне таблицы.

Удалить представление может только владелец представления, администратор, пользователь с привилегией `DROP ANY VIEW`.

15.5 Пересоздание представлений

Оператор `RECREATE VIEW` позволяет внести изменения в существующее представление.

Листинг 15.5. Синтаксис оператора пересоздания представления `RECREATE VIEW`

```
RECREATE VIEW <имя представления> [<список столбцов>]
AS <оператор SELECT>
[WITH CHECK OPTION]

<список столбцов> ::= (столбец [, столбец ...])
```

Представление может отсутствовать в базе данных. В этом случае оно просто создается заново. Если представление с этим именем уже существует в базе данных, то оно удаляется и затем создается заново. Попытка выполнить пересоздание представления приведет к ошибке базы данных, если это представление в настоящий момент находится в использовании. Оператор `RECREATE VIEW` не выполнится, если существующее представление имеет зависимости.

15.6 Примеры представлений

Любой пример оператора SELECT из главы 11 можно записать в виде представления.

Пример 1.

В одном из примеров главы 11 был приведен пример использования производной таблицы. Можно создать соответствующее представление USER_TABLES. Это представление отображает все пользовательские таблицы базы данных.

```
CREATE VIEW USER_TABLES
   (RDB$RELATION_NAME, RDB$RELATION_ID)
AS
   SELECT *
   FROM (SELECT
            RDB$RELATION_NAME,
            RDB$RELATION_ID
         FROM RDB$RELATIONS
         WHERE RDB$RELATION_NAME NOT STARTING WITH 'RDB$')
   AS R ("Таблица", "Идентификатор");
```

Здесь выбираются таблицы, чьи имена не начинаются с символов 'RDB$', то есть таблицы, не являющиеся системными. В этом представлении должны быть обязательно указаны имена столбцов после имени представления. В примере заданы те же имена столбцов, что и в операторе SELECT. Фактически имена могут быть любыми. Эти имена можно использовать в операторе SELECT, который обращается к данному представлению. Оператор, выполняющий данное представление:

```
SELECT RDB$RELATION_NAME, RDB$RELATION_ID
FROM USER_TABLES;
```

Во многих случаях представление бывает особенно полезным тогда, когда оператор выборки данных SELECT является довольно сложным. Использование представлений позволит сократить объем ручного ввода пользователем и уменьшить вероятность ошибок.

Пример 2.

Можно составить представление для получения списка людей и их родителей, что было показано в операторе SELECT в примере в главе 11. Соответствующий оператор SELECT представления также содержит два левых внешних соединения с той же таблицей.

```
CREATE VIEW SELECT_PEOPLE (C1, C2, C3)
AS
  SELECT
      PG.FULLNAME AS "Фамилия, имя, отчество",
      PM.NAME3 AS "Мать",
      PF.NAME3 AS "Отец"
  FROM PEOPLE PG             /* Главная таблица */
      LEFT OUTER JOIN PEOPLE PM    /* Мать */
         ON PG.CODMOTHER = PM.COD
      LEFT OUTER JOIN PEOPLE PF    /* Отец */
         ON PG.CODFATHER = PF.COD;
```

Это представление не является естественно изменяемым, поскольку список выбора не содержит столбец код человека, а этот столбец, являясь первичным ключом таблицы, не допускает пустого значения NULL. По этой причине такое представление вообще не может быть сделано изменяемым даже и при помощи триггеров (см. далее в разделе 15.7 этой главы). Кроме того, в представлении выполняется

соединение таблиц (для определения, является ли представление естественно изменяемым, неважно, что таблица соединяется сама с собой; проблема не в том, что при изменении затрагивается не одна, а несколько таблиц, а в том, что изменение происходит для нескольких строк). Попытки выполнить добавление новых данных, изменение или удаление существующих строк приведут к выдаче диагностического сообщения о попытке изменять представление только для чтения (`read-only`).

Список имен столбцов в самом представлении (`C1, C2, C3`) никак не связан с именами столбцов, получаемых из базовых таблиц, что вполне допустимо.

Для выборки данных при помощи этого представления можно использовать, например, следующий оператор `SELECT`:

```
SELECT
    C1 AS "Фамилия, имя, отчество",
    C2 AS "Мать",
    C3 AS "Отец"
FROM SELECT_PEOPLE
ORDER BY C1;
```

Пример 3.

Следующий несколько надуманный пример демонстрирует использование предложения `WITH CHECK OPTION`. Представление является естественно изменяемым, потому что данные выбираются из одной таблицы, и единственный столбец, который не допускает пустого значения `NULL` (первичный ключ, код страны — `CODCOUNTRY`) включен в список выбора.

```
CREATE VIEW V_TEST (CODCOUNTRY, NAME, FULLNAME, CAPITAL, DESCR)
AS
  SELECT *
  FROM COUNTRY
  WHERE CODCOUNTRY CONTAINING 'A'
WITH CHECK OPTION;
```

Здесь выбираются все страны из таблицы стран, у которых в коде страны присутствует символ «A» — неважно, в начале, в середине или в конце строки. Предложение `WITH CHECK OPTION` задает такое условие, что пользователь не может добавить новую запись в полученный набор данных или изменить код страны существующей записи, если не будет выполняться условие в предложении `WHERE`, то есть, если код страны не будет содержать символа «A».

Пример 4.

В следующем простом представлении выбираются все регионы всех стран, хранящиеся в базе данных.

```
CREATE VIEW V_REGION (CODCOUNTRY, CODREGION, NAMEREG, CENTER, DESCR)
AS
  SELECT * FROM REGION;
```

Это представление является естественно изменяемым. Для него можно выполнять операторы добавления, изменения и удаления существующих данных, как если бы это представление было обычной таблицей. Нет необходимости в написании триггеров для внесения изменений в базовую таблицу.

Пример 5.

С таблицей, получаемой при обращении с помощью оператора `SELECT` к представлению, можно выполнять все действия, как и с обычной таблицей. В частности, можно выполнить соединение полученной при обращении к представлению таблицы с другой таблицей базы данных. Следующий оператор

SELECT, при обращении к только что описанному представлению, выполняет соединение каждой строки из списка регионов со справочником стран, добавляя в результирующий набор данных краткое и полное название страны, к которой относится выбираемый в операторе регион:

```
SELECT
  V_REGION.*,
  COUNTRY.NAME,
  COUNTRY.FULLNAME
FROM V_REGION
  LEFT OUTER JOIN COUNTRY
      ON V_REGION.CODCOUNTRY = COUNTRY.CODCOUNTRY
ORDER BY 1;
```

Пример 6.

Можно создать естественно изменяемое представление VIEW_RUSSIA2, которое выбирает все регионы России. Оператор создания такого представления показан в следующем примере. Здесь код страны 'РОССИЯ' получается при помощи внутреннего оператора SELECT, который получает код на основании заданного краткого названия страны.

```
CREATE VIEW VIEW_RUSSIA2 (CODCOUNTRY, CODREGION, NAMEREG, CENTER)
AS
  SELECT
     CODCOUNTRY, CODREGION, NAMEREG, CENTER
FROM REGION
WHERE CODCOUNTRY = (SELECT CODCOUNTRY
                    FROM COUNTRY
                    WHERE NAME = 'Россия');
```

Просмотреть регионы с использованием данного представления можно обычным оператором SELECT:

```
SELECT
   CODCOUNTRY  AS "Код страны",
   CODREGION  AS "Код региона",
   NAMEREG  AS "Название региона",
   CENTER  AS "Центр региона"
FROM VIEW_RUSSIA2;
```

При использовании этого представления можно изменить значение любого столбца базовой таблицы (изменять можно значения только тех столбцов, которые явно описаны в этом представлении). Например, можно изменить название региона:

```
UPDATE VIEW_RUSSIA2
   SET NAMEREG = 'Неизвестная область'
WHERE CODREGION = '64' AND
      CODCOUNTRY = (SELECT CODCOUNTRY
                    FROM COUNTRY
                    WHERE NAME = 'Россия');
```

Разумеется, при использовании данного представления можно изменять и код региона, и название центра региона. Здесь можно также изменить и код страны, однако с учетом условия выборки данных регионов при помощи данного представления, такая строка не будет отображена при дальнейшем обращении к этому представлению.

Используя данное представление, можно добавить новый регион. При этом обязательно нужно

указать значения всех столбцов, входящих в состав первичного ключа, несмотря на то, что в представлении как бы уже неявно задано условие, что значением кода страны является код России.

```
INSERT INTO VIEW_RUSSIA2 (CODCOUNTRY, CODREGION, NAMEREG)
VALUES ((SELECT CODCOUNTRY
          FROM COUNTRY
          WHERE NAME = 'Россия'), '00', 'Несуществующий регион');
```

Используя данное естественно изменяемое представление, можно удалять строки из базовой таблицы. Здесь удаляется строка, помещенная в таблицу в предыдущем примере.

```
DELETE FROM VIEW_RUSSIA2
WHERE CODREGION = '00' AND
      CODCOUNTRY = (SELECT CODCOUNTRY
                    FROM COUNTRY
                    WHERE NAME = 'Россия');
```

15.7 Преобразование неизменяемых представлений в изменяемые при помощи триггеров

Как уже было сказано, многие представления, являющиеся неизменяемыми (`read-only`), могут быть сделаны изменяемыми при создании соответствующих триггеров. Такое преобразование возможно только в том случае, если все столбцы, не находящиеся в списке выбора при выполнении обращения к данному представлению, могут принимать пустое значение `NULL`. Иными словами, все столбцы, входящие в состав первичного ключа, и все остальные столбцы, для которых явно задано условие `NOT NULL`, должны присутствовать в списке выбора оператора `SELECT` в представлении.

Для того чтобы при использовании представления можно было выполнять оператор добавления данных (`INSERT`), необходимо для этого представления написать триггер, выполняемый до добавления (`BEFORE INSERT`). Чтобы к представлению можно было применять операцию изменения данных, необходим триггер `BEFORE UPDATE`, операцию удаления — триггер `BEFORE DELETE`.

Если представление является естественно изменяемым и для него созданы соответствующие триггеры, то любая операция изменения не будет выполнять прямые изменения в таблице. Все изменения будут выполняться только в триггерах. Если триггеры фактически не содержат операторов DML по изменению данных в таблице, то никакие изменения не будут произведены.

Следующий пример создает представление `V_REGION_COUNTRY`, которое не является естественно изменяемым, потому что содержит соединение (`JOIN`) двух таблиц. При этом представление включает в себя все столбцы базовых таблиц, которые не могут принимать пустые значения. Следовательно, существует принципиальная возможность сделать это представление изменяемым.

```
CREATE VIEW V_REGION_COUNTRY
   (CODCOUNTRY, CODREGION, NAMEREG, CENTER, DESCR, NAME, FULLNAME)
AS
   SELECT
      REGION.CODCOUNTRY,
      CODREGION,
      NAMEREG,
      CENTER,
      REGION.DESCR,
      COUNTRY.NAME,
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
      COUNTRY.FULLNAME
   FROM REGION
      LEFT OUTER JOIN COUNTRY
         ON REGION.CODCOUNTRY = COUNTRY.CODCOUNTRY;
```

В этом представлении отыскиваются все регионы всех стран, выбираются все столбцы из таблицы регионов и для каждой строки добавляется краткое (NAME) и полное (FULLNAME) название соответствующей страны.

Это представление не является естественно изменяемым, для него нельзя без дополнительных действий использовать операторы INSERT, UPDATE или DELETE. Поскольку это представление включает в себя все столбцы, которые не могут иметь пустого значения NULL, то такое представление легко можно сделать изменяемым, создав для него соответствующие вспомогательные триггеры. Все триггеры должны выполняться до (BEFORE) соответствующего действия — добавления, изменения, удаления.

Чтобы предоставить пользователю системы возможность добавлять в базовые таблицы этого представления новые строки, необходимо для представления создать только лишь триггер, который будет вызываться до добавления строки (BEFORE INSERT). При наличии такого триггера можно добавлять новые строки в базовые таблицы, но при наличии только этого триггера выполнять иные действия по изменению и удалению будет невозможно.

Будет ли на самом деле реально добавлена новая строка в одну таблицу или несколько строк в обе (во все, используемые в операторе SELECT) базовые таблицы при использовании для этого представления оператора INSERT зависит от того, какой код содержится в самом триггере. В триггере может быть задано добавление новых строк в обе (во все) или только в одну базовую таблицу. Триггер также может вообще не содержать операторов добавления новых строк. При этом в случае наличия соответствующего триггера для данного представления выполнение оператора INSERT для такого представления не вызовет исключения или ошибок базы данных (поскольку просто лишь существует подходящий триггер BEFORE INSERT). Пример такого триггера для представления V_REGION_COUNTRY приведен в ниже. У пользователя, выполняющего такой оператор INSERT, может сложиться впечатление, что все действия привели к нужным ему результатам, что не всегда соответствует действительности. Это может стать хорошим источником всевозможных ошибок.

```
SET TERM ^;
CREATE TRIGGER TBI_REGION_COUNTRY
   FOR V_REGION_COUNTRY
   BEFORE INSERT
AS
BEGIN
END ^
```

Чтобы дать еще и возможность при использовании описанного представления выполнять также и изменения в одной из базовых таблиц, а именно внесение изменений в справочник регионов REGION, нужно создать триггер, который будет выполняться до изменения (BEFORE UPDATE) этого представления.

```
SET TERM ^;
CREATE TRIGGER TBU_REGION_COUNTRY
   FOR V_REGION_COUNTRY
   BEFORE UPDATE
AS
BEGIN
    UPDATE REGION
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
    SET NAMEREG = NEW.NAMEREG
    WHERE CODCOUNTRY = OLD.CODCOUNTRY AND CODREGION = OLD.CODREGION;
END ^
```

Этот триггер позволяет изменять только название региона и только в таблице регионов. Другие столбцы этой таблицы, а также любые столбцы соединяемой родительской таблицы стран изменяться не будут. Например, следующий оператор будет выполнен и выполнен правильно.

```
UPDATE V_REGION_COUNTRY
SET NAMEREG = 'Another region'
WHERE CODREGION = '64';
```

Следующий оператор также не вызовет никаких сообщений об ошибках, однако его выполнение не приведет ни к каким изменениям в базовых таблицах.

```
UPDATE V_REGION_COUNTRY
SET FULLNAME = 'Еще одна страна'
WHERE CODREGION = '64';
```

Здесь предполагается, что должно быть изменено полное название страны в таблице стран, которая является родительской для указанной строки регионов. Однако, поскольку такое изменение никак не описано в триггере, то никакие действия по модификации данных выполнены не будут.

Следующий триггер выполняет удаление заданных строк из обеих базовых таблиц — как из таблицы регионов, так и из таблицы стран. Триггер должен выполняться до удаления (`BEFORE DELETE`) данного представления.

```
SET TERM ^;
CREATE TRIGGER TBD_REGION_COUNTRY
   FOR V_REGION_COUNTRY
   BEFORE DELETE
AS
BEGIN
    DELETE FROM REGION
    WHERE CODCOUNTRY = OLD.CODCOUNTRY AND CODREGION = OLD.CODREGION;
    DELETE FROM COUNTRY
    WHERE CODCOUNTRY = OLD.CODCOUNTRY;
END ^
```

Следующий оператор выполняет удаление заданных строк в обеих базовых таблицах:

```
DELETE FROM V_REGION_COUNTRY
WHERE CODCOUNTRY = 'USA';
```

Вообще говоря, удаление в данном триггере не только страны, но еще и региона явно является излишним, так как удаление заданной строки страны автоматически приведет к удалению всех регионов этой страны (если в описании внешнего ключа в дочерней таблице регионов задано `ON DELETE CASCADE`).

Вместо написания трех триггеров можно создать один для всех обновляющих действий фазы `BEFORE`. Пример триггера, объединяющего все функции предыдущих триггеров:

```
SET TERM ^;
CREATE TRIGGER TBC_REGION_COUNTRY
   FOR V_REGION_COUNTRY
   BEFORE UPDATE OR INSERT OR DELETE
AS BEGIN
   IF (UPDATING) THEN
   BEGIN
      UPDATE REGION
      SET NAMEREG = NEW.NAMEREG
      WHERE CODCOUNTRY = OLD.CODCOUNTRY AND CODREGION = OLD.CODREGION;
   END
   IF (INSERTING) THEN
   BEGIN
   END
   IF (DELETING) THEN
   BEGIN
      DELETE FROM REGION
      WHERE CODCOUNTRY = OLD.CODCOUNTRY AND CODREGION = OLD.CODREGION;
      DELETE FROM COUNTRY
      WHERE CODCOUNTRY = OLD.CODCOUNTRY;
   END
END ^
```

В этом триггере объединяется функциональность трех предыдущих триггеров, относящихся к представлению `V_REGION_COUNTRY`.

С целью определения, для какой обновляющей операции вызывается триггер, используются контекстные переменные `UPDATING, INSERTING` и `DELETING`. Подробнее о контекстных переменных см. в главе 20. В этом триггере также не осуществляется никаких действий по добавлению данных, однако выполнение оператора `INSERT` для представления `V_REGION_COUNTRY` не вызовет ошибок.

Если и для базовых таблиц представления также заданы соответствующие триггеры для фаз `BEFORE`, то выполнение триггеров для представления происходит прежде, чем выполнение этих триггеров для базовых таблиц. Это нужно учитывать, в частности и при формировании значения искусственного первичного ключа, которое выбирается из генератора (`GENERATOR, SEQUENCE`).

Преобразование неизменяемых представлений в изменяемые при помощи триггеров требует особой, повышенной осторожности.

Подробности о языке хранимых процедур и триггеров (PSQL), о создании, удалении и изменении триггеров см. в главе 17 и главе 20.

15.8 Системные представления

В стандарте SQL-92 определены представления для отображения системных сведений об ограничениях целостности в базе данных. Здесь приведено четыре полезных системных представления.

Пример 1.

Представление `CHECK_CONSTRAINTS`, показанное в следующем примере, позволяет получить список всех ограничений базы данных — ограничений первичного, уникального, внешнего ключей и ограничений `CHECK`. В случае ограничения `CHECK` вторым столбцом выводится текст условия ограничения.

```
CREATE VIEW CHECK_CONSTRAINTS (CONSTRAINT_NAME, CHECK_CLAUSE)
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
AS
   SELECT RDB$CONSTRAINT_NAME, RDB$TRIGGER_SOURCE
   FROM RDB$CHECK_CONSTRAINTS RC, RDB$TRIGGERS RT
   WHERE RT.RDB$TRIGGER_NAME = RC.RDB$TRIGGER_NAME;
```

Для обращения к этому представлению можно использовать, например, такой оператор SELECT:

```
SELECT
   CONSTRAINT_NAME AS "Имя ограничения",
   CHECK_CLAUSE AS "Предложение CHECK"
FROM CHECK_CONSTRAINTS;
```

Пример 2.

Представление CONSTRAINTS_COLUMN_USAGE выводит список всех таблиц базы данных, их столбцов и ограничений, применяющихся для соответствующих столбцов. Здесь отображаются только сведения о первичных, уникальных и внешних ключах. Ограничения CHECK в этом списке отсутствуют.

```
CREATE VIEW CONSTRAINTS_COLUMN_USAGE
   (TABLE_NAME, COLUMN_NAME, CONSTRAINT_NAME)
AS
  SELECT
      RDB$RELATION_NAME,
      RDB$FIELD_NAME,
      RDB$CONSTRAINT_NAME
  FROM RDB$RELATION_CONSTRAINTS RC, RDB$INDEX_SEGMENTS RI
  WHERE RI.RDB$INDEX_NAME = RC.RDB$INDEX_NAME;
```

Получить соответствующий список можно, выполнив, например, следующий оператор SELECT:

```
SELECT
   TABLE_NAME AS "Таблица",
   COLUMN_NAME AS "Столбец",
   CONSTRAINT_NAME AS "Имя ограничения"
FROM CONSTRAINTS_COLUMN_USAGE;
```

Пример 3.

В следующем примере показано системное представление, отображающее список всех внешних ключей в базе данных. В каждой строке в первом столбце содержится имя таблицы, затем имя внешнего ключа, после этого имя первичного или уникального ключа, на который ссылается данный внешний ключ, потом уровень соответствия (в данной версии всегда FULL — полное соответствие внешнего ключа ключу родительской таблицы; будет использовано в дальнейших версиях), поведение системы при изменении значения (ON UPDATE) ключевого реквизита родительской таблицы и при удалении строки (ON DELETE) родительской таблицы.

```
CREATE VIEW REFERENTIAL_CONSTRAINTS
   (CONSTRAINT_NAME, UNIQUE_CONSTRAINT_NAME, MATCH_OPTION,
    UPDATE_RULE, DELETE_RULE)
AS
  SELECT
      RDB$CONSTRAINT_NAME,
      RDB$CONST_NAME_UQ,
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
       RDB$MATCH_OPTION,
       RDB$UPDATE_RULE,
       RDB$DELETE_RULE
   FROM RDB$REF_CONSTRAINTS;
```

Для получения списка ограничений внешнего ключа базы данных можно использовать следующий оператор SELECT:

```
SELECT
   CONSTRAINT_NAME AS "Внешний ключ",
   UNIQUE_CONSTRAINT_NAME AS "Первичный/уникальный ключ",
   MATCH_OPTION AS "Уровень соответствия",
   UPDATE_RULE AS "ON UPDATE",
   DELETE_RULE AS "ON DELETE"
FROM REFERENTIAL_CONSTRAINTS;
```

Пример 4.

Представление TABLE_CONSTRAINTS позволяет получить список всех ограничений всех таблиц. Сюда входят ограничения первичного, уникального, внешнего ключей, ограничение CHECK и ограничение NOT NULL. Каждая строка содержит имя ограничения (имя, заданное пользователем при описании ограничения, или имя, сгенерированное системой, если пользователь не задал никакого имени), имя таблицы, для которой создано ограничение, и вид ограничения. Последние два символьных столбца, указанные в этом представлении (IS_DEFERRABLE и INITIALLY_DEFERRED), во всех строках содержат текст "NO", они будут использованы в дальнейших расширениях системы.

```
CREATE VIEW TABLE_CONSTRAINTS
   (CONSTRAINT_NAME, TABLE_NAME, CONSTRAINT_TYPE,
    IS_DEFERRABLE, INITIALLY_DEFERRED)
AS
  SELECT
      RDB$CONSTRAINT_NAME,
      RDB$RELATION_NAME,
      RDB$CONSTRAINT_TYPE,
      RDB$DEFERRABLE,
      RDB$INITIALLY_DEFERRED
  FROM RDB$RELATION_CONSTRAINTS;
```

Обращение к этому представлению может быть выполнено следующим оператором SELECT:

```
SELECT
   CONSTRAINT_NAME AS "Имя ограничения",
   TABLE_NAME AS "Таблица",
   CONSTRAINT_TYPE AS "Вид ограничения",
   IS_DEFERRABLE,
   INITIALLY_DEFERRED
FROM TABLE_CONSTRAINTS;
```

15.9 Примечание представления

Для представления можно создать примечание, используя следующий синтаксис оператора COMMENT:

Листинг 15.6. Синтаксис оператора примечания представления COMMENT ON VIEW

```
COMMENT ON VIEW <имя представления> IS {'<текст>' | NULL};
```

Если в качестве текста примечания задать NULL, то будет удалено существующее примечание представления.

Пример.

Чтобы добавить примечание к представлению V_REGION, нужно выполнить оператор:

```
COMMENT ON VIEW V_REGION IS 'Выбор всех регионов всех стран';
```

Глава 16

Исключения (EXCEPTION)

Пользовательское исключение (exception) — объект базы данных, описывающий сообщение об ошибке. Исключение можно вызывать и обрабатывать в PSQL коде.

16.1 Создание исключений

Для создания пользовательского исключения используется оператор CREATE EXCEPTION. Его синтаксис показан в листинге 16.1.

Листинг 16.1. Синтаксис оператора создания пользовательского исключения CREATE EXCEPTION

```
CREATE EXCEPTION <имя исключения> '<текст сообщения>';
```

Имя исключения может содержать до 63 символов и должно быть уникальным среди всех имен пользовательских исключений базы данных. Имя исключения является стандартным идентификатором. В диалекте 3 оно может быть заключено в двойные кавычки, что делает его чувствительным к регистру.

Текст сообщения — это текст, выдаваемый пользователю при вызове данного исключения. Может содержать до 1021 любых символов, включая буквы кириллицы и специальные символы. Сообщение об ошибке может содержать слоты для параметров (@N), которые заполняются при возбуждении исключения. Нумерация слотов начинается с 1. Максимальный номер слота равен 9.

Создать исключение может администратор и пользователь с привилегией CREATE EXCEPTION.

Пользователь, создавший исключение, становится его владельцем.

```
CREATE EXCEPTION E_LARGE_VALUE 'Значение превышает предельно допустимое';
```

```
CREATE EXCEPTION E_INVALID_VALUE 'Неверное значение @1 для поля @2';
```

Более подробно о пользовательских исключениях можно ознакомиться в главе 17.5.

16.2 Изменение исключений

Для изменения текста сообщения существующего пользовательского исключения используется оператор ALTER EXCEPTION. Синтаксис оператора см. в листинге 16.2:

Листинг 16.2. Синтаксис оператора изменения пользовательского исключения ALTER EXCEPTION

```
ALTER EXCEPTION <имя исключения> '<текст сообщения>';
```

Изменять текст сообщения пользовательского исключения может администратор, владелец исключения и пользователь с привилегией ALTER ANY EXCEPTION.

16.3 Создание или изменение исключений

Оператор CREATE OR ALTER EXCEPTION создает новое пользовательское исключение, если оно отсутствует в базе данных, или изменяет существующее, при этом существующие зависимости исключения будут сохранены. Синтаксис оператора представлен в листинге 16.3.

Листинг 16.3. Синтаксис оператора CREATE OR ALTER EXCEPTION

```
CREATE OR ALTER EXCEPTION <имя исключения> '<текст сообщения>';
```

16.4 Пересоздание исключений

Оператор RECREATE EXCEPTION создает новое пользовательское исключение, если оно отсутствует в базе данных, или пересоздает существующее. Синтаксис оператора см. в листинге 16.4.

Листинг 16.4. Синтаксис оператора RECREATE EXCEPTION

```
RECREATE EXCEPTION <имя исключения> '<текст сообщения>';
```

16.5 Удаление исключений

Для удаления существующего пользовательского исключения используется оператор DROP EXCEPTION.

Листинг 16.5. Синтаксис оператора удаления пользовательского исключения DROP EXCEPTION

```
DROP EXCEPTION <имя исключения>;
```

Удалить пользовательское исключение может администратор, владелец исключения или пользователь с привилегией DROP ANY EXCEPTION.

Пользовательское исключение нельзя удалить, если оно используется в операторе EXCEPTION в каком-либо триггере или хранимой процедуре и функции.

Глава 17

Процедурный язык PSQL

Procedural SQL (PSQL) - процедурное расширение языка SQL. Это подмножество языка используется для написания хранимых процедур, хранимых функций, пакетов, триггеров и PSQL блоков.

Это расширение содержит все основные конструкции классических языков программирования. Кроме того, в него входят немного модифицированные DML операторы (SELECT, INSERT, UPDATE, DELETE и др.).

17.1 Элементы PSQL

Процедурное расширение может содержать объявления локальных переменных и курсоров, операторы присваивания, условные операторы, операторы циклов, выброса пользовательского исключений, средства для обработки ошибок, отправки сообщений (событий) клиентским программам. Кроме того, в триггерах доступны специфичные контекстные переменные, такие как NEW и OLD.

В PSQL не допустимы операторы модификации метаданных (DDL операторы).

17.1.1 DML операторы с параметрами

В DML (SELECT, INSERT, UPDATE, DELETE и др.) операторах допустимы только именованные параметры. Если DML операторы содержат именованные параметры, то они должны быть предварительно объявлены как локальные переменные в операторе DECLARE [VARIABLE] заголовка модуля или доступны во входных или выходных параметрах PSQL модуля.

При использовании именованных параметров в DML операторах необходим префикс двоеточия ":", однако в предложении INTO символ двоеточия не обязателен. Префикс двоеточия является необязательным в операторах специфичных для PSQL, таких, как операторы ветвления или присваивания. Префикс двоеточия не требуется также при вызове хранимой процедуры с помощью оператора EXECUTE PROCEDURE из другого PSQL модуля.

17.1.2 Транзакции

Хранимые процедуры и функции, в том числе содержащиеся в пакетах, выполняются в контексте той транзакции, в которой они были запущены. Триггеры выполняются в контексте транзакции, в которой выполнялся DML оператор, вызвавший запуск триггера. Для триггеров на событие базы данных запускается отдельная транзакция.

В PSQL не допустимы операторы старта и завершения транзакций, но существует возможность запуска оператора или блока операторов в автономной транзакции.

17.1.3 Структура модуля

В синтаксисе PSQL модулей можно выделить заголовок и тело. DDL операторы для их объявления являются сложными операторами, т.е. состоят из единственного оператора, который включает в себя блоки нескольких операторов. Такие операторы начинаются с глагола (CREATE, ALTER, DROP, RECREATE, CREATE OR ALTER) и завершаются последним оператором END тела модуля.

Заголовок модуля

Заголовок содержит имя модуля и описание локальных переменных. Для хранимых процедур и PSQL блоков заголовок может содержать описание входных и выходных параметров. Заголовок триггеров не может содержать входных и выходных параметров.

В заголовке триггера обязательно указывается событие (или комбинация событий), при отором триггер будет вызван автоматически.

Привилегии выполнения PSQL кода

PSQL код может выполняться в одном из следующих режимов:

- С привилегиями вызывающего пользователя (привилегии CURRENT_USER);
- С привилегиями определяющего пользователя (владельца объекта метаданных).

Привилегии выполнения PSQL модуля указывается в его заголовке в необязательное предложение SQL SECURITY. Если выбрана опция INVOKER, то PSQL модуль выполняются с привилегиями вызывающего пользователя. Если выбрана опция DEFINER, то PSQL модуль выполняется с привилегиями определяющего пользователя (владельца). Эти привилегии будут дополнены привилегиями выданные самому PSQL модулю с помощью оператора GRANT. По умолчанию процедуры, функции выполняются с привилегиями вызывающего пользователя, а триггеры наследуют привилегии безопасности указанные для таблицы.

Анонимные PSQL блоки (EXECUTE BLOCK) всегда выполняются с правами вызывающего пользователя.

Тело модуля

Тело модуля может быть написано на языке PSQL или быть телом внешнего модуля.

```
<определение тела модуля> ::=
  <определение модуля>
| <определение внешнего модуля>

<определение модуля> ::=
[<привилегии выполнения>] <тело модуля>

<привилегии выполнения> ::=
SQL SECURITY {DEFINER | INVOKER}

<тело модуля> ::=
AS
        [<объявление>]
BEGIN
        [<блок операторов>]
END

<объявление> ::=
<объявление элемента> [<объявление элемента> ...]

<объявление элемента> ::=
  <объявление локальной переменной>;
| <объявление именованного курсора>;
| <объявление подпрограммы>;
| <реализация подпрограммы>
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
<объявление подпрограммы> ::= <объявление подфункции> | <объявление подпроцедуры>

<определение внешнего модуля> ::=
EXTERNAL NAME <внешний модуль> ENGINE <имя движка>
[AS <тело внешнего модуля>]

<реализация подпрограммы> ::= <реализация подфункции> | <реализация подпроцедуры>

<внешний модуль> ::= '<имя внешнего модуля>!<имя функции в модуле>[! <информация>]'
```

Таблица 17.1 — Параметры тела модуля

Параметр	Описание
Тело внешнего модуля	Тело внешнего модуля. Строковый литерал который может использоваться UDR для различных целей.
Имя внешнего модуля	Имя внешнего модуля, в котором расположена функция.
Имя функции в модуле	Внутреннее имя функции внутри внешнего модуля.
Информация	Определяемая пользователем информация для передачи в функцию внешнего модуля.
Имя движка	Имя движка для использования внешних функций. Обычно указывается имя UDR.

Тело PSQL модуля

Тело PSQL начинается с необязательного раздела, в котором объявляются переменные, курсоры и подпрограммы. Далее следует блок операторов, которые выполняются в логической последовательности как программа. Блок операторов - или составной оператор - заключен в ключевые слова BEGIN и END и выполняется как единый блок кода. Основной блок BEGIN ... END может содержать любое количество других блоков BEGIN ... END, как встроенных, так и последовательных. Максимальная вложенность блоков составляет 512 уровней.

Оператор SET TERM

Символом конца строки (завершения команды) по умолчанию является точка с запятой. Этот символ можно изменить командой:

```
SET TERM <строка>;
```

где <строка> может быть как одним символом, так и группой символов.

При написании триггеров и хранимых процедур в текстах скриптов, создающих требуемые программные объекты базы данных, во избежание двусмысленности относительно использования символа завершения операторов (по нормам SQL это точка с запятой) применяется оператор SET TERM, который, строго говоря, не является оператором SQL. При помощи этого псевдооператора перед началом создания триггера, хранимой функции или хранимой процедуры задается символ, являющийся завершающим. После описания текста соответствующего программного объекта при помощи того же оператора SET TERM значение терминатора возвращается к обычному варианту — точка с запятой.

Например, при создании некоторого триггера, хранимой функции или хранимой процедуры следует выполнить следующие операторы:

```
SET TERM ^;
/* Формирование значение первичного ключа таблицы STAFF */
CREATE TRIGGER TBI_STAFF FOR STAFF
   BEFORE INSERT
AS
BEGIN
   ...           /*Текст триггера*/
END ^
SET TERM ;^
```

Здесь вначале в качестве терминатора выбирается символ ^ (это наиболее часто выбираемый символ, поскольку он довольно редко присутствует в программных текстах; в SQL он используется в качестве символа отрицания, который можно представить и восклицательным знаком). После операторов создания триггеров, функций и хранимых процедур терминатор возвращается в исходное состояние.

Тело внешнего модуля

Тело внешнего модуля определяет механизм UDR, используемый для выполнения внешнего модуля, и дополнительно указывает имя вызываемой процедуры UDR (<внешний модуль>) и/или строку (<тело внешнего модуля>) с семантикой, специфичной для UDR.

Конфигурация внешних модулей и механизмов UDR не рассматривается далее в этом справочнике по языку. За подробностями обращайтесь к документации по конкретному движку UDR.

17.2 PSQL блоки

Для выполнения из декларативного SQL (DSQL) некоторых императивных действий используются анонимные (безымянные) PSQL блоки. Заголовок анонимного PSQL блока опциально может содержать входные и выходные параметры. Тело анонимного PSQL блока может содержать объявление локальных переменных, курсоров, подпрограмм и блок PSQL операторов.

Анонимный PSQL блок не определяется и сохраняется как объект метаданных, в отличие от хранимых процедур и триггеров. Он не может обращаться сам к себе.

Как и хранимые процедуры анонимные PSQL блоки могут использоваться для обработки данных или для осуществления выборки из базы данных.

Синтаксис блока PSQL:

```
EXECUTE BLOCK
        [(<входной параметр> = ? [, <входной параметр> = ? ...])]
        [RETURNS (<выходной параметр> [, <выходной параметр> ...])]
        <определение тела модуля>
```

Подробное описание EXECUTE BLOCK см. в листинге 11.31.

17.3 Написание кода тела модуля

В этом разделе подробно рассматривается процедурные конструкции языка SQL и операторы доступные в теле хранимых процедур, триггеров и анонимных PSQL блоков.

Маркер двоеточия

Маркер двоеточия (:) используется в PSQL, чтобы пометить ссылку на переменную в DML операторе. В остальных случаях маркер двоеточия необязателен перед именами переменных.

Никогда не задавайте префикс двоеточия для контекстных переменных.

17.3.1 Оператор присваивания

Присваивание переменной значения.

Синтаксис операции присваивания значения внутренней переменной имеет следующий вид:

```
<имя переменной> = <выражение>;
```

- Имя переменной - Имя локальной переменной или параметра процедуры (функции).
- Выражение - Выражение, константа или переменная совместимая по типу данных.

PSQL использует символ равенства (=) в качестве своего оператора присваивания. Оператор присваивания устанавливает переменной слева от оператора значение SQL выражения справа. Выражением может быть любое правильное выражение SQL. Оно может содержать литералы, имена внутренних переменных, арифметические, логические и строковые операции, обращения к встроенным функциям и к функциям, определённым пользователем.

Пример использования оператора присваивания:

```
CREATE PROCEDURE MYPROC (
        a INTEGER,
        b INTEGER,
        name VARCHAR (30)
)
RETURNS (
        c INTEGER,
        str VARCHAR(100))
AS
BEGIN
        -- присваиваем константу
        c = 0;
        str = '';
        SUSPEND;
        -- присваиваем значения выражений
        c = a + b;
        str = name || CAST(b AS VARCHAR(10));
        SUSPEND;
        -- присваиваем значение выражения
        -- построенного с использованием запроса
        c = (SELECT 1 FROM rdb$database);
        -- присваиваем значение из контекстной переменной
        str = CURRENT_USER;
        SUSPEND;
END
```

17.3.2 Объявление локальной переменной

Для описания одной локальной переменной используется оператор DECLARE VARIABLE. Синтаксис оператора представлен в листинге 17.1.

Листинг 17.1. Синтаксис оператора объявления локальной переменной и курсора DECLARE VARIABLE

```
DECLARE [VARIABLE] {
   <имя локальной переменной> <тип>
   [NOT NULL]
   [COLLATE <порядок сортировки>]
   [{ = | DEFAULT } <значение по умолчанию>]
}

<тип> ::= {
   <тип данных SQL>
 | [TYPE OF] <имя домена>
 | TYPE OF COLUMN <имя таблицы/представления>.<имя столбца> }

<значение по умолчанию>::= {<литерал> | NULL | <контекстная переменная>}
```

Оператор DECLARE [VARIABLE] объявляет локальную переменную. Ключевое слово VARIABLE можно опустить. В одном операторе разрешено объявлять только одну переменную. В процедурах и триггерах можно объявить произвольное число локальных переменных, используя при этом каждый раз, новый оператор DECLARE VARIABLE.

Имя локальной переменной должно быть уникально среди имён локальных переменных, входных и выходных параметров процедуры внутри программного объекта.

Типы данных для переменных

В качестве типа данных локальной переменной может быть любой SQL тип, за исключением массивов.

В качестве типа переменной можно указать имя домена. В этом случае переменная будет наследовать все характеристики домена. Если перед названием домена дополнительно используется предложение TYPE OF, то используется только тип данных домена - не проверяется (не используется) его ограничение (если оно есть в домене) на NOT NULL, CHECK ограничения и/или значения по умолчанию. Если домен текстового типа, то всегда используется его набор символов и порядок сортировки.

Локальные переменные можно объявлять, используя тип данных столбцов существующих таблиц и представлений. Для этого используется предложение TYPE OF COLUMN, после которого указывается имя таблиц или представления и через точку имя столбца. При использовании TYPE OF COLUMN наследуется только тип данных, а в случае строковых типов ещё набор символов и порядок сортировки. Ограничения и значения по умолчанию столбца никогда не используются.

Ограничение NOT NULL

Для локальных переменных можно указать ограничение NOT NULL, тем самым запретив передавать в него значение NULL.

Предложения CHARACTER SET и COLLATE

Если не указано иное, набор символов и последовательность сопоставления (сортировки) строковой переменной будут значениями по умолчанию для базы данных. При необходимости можно включить предложение CHARACTER SET для обработки строковых данных, которые будут находиться в другом наборе символов. Допустимая последовательность сопоставления (предложение COLLATE) также может быть включена с CHARACTER SET или без него.

Инициализация переменной

Локальной переменной можно устанавливать инициализирующее (начальное) значение. Это значение устанавливается с помощью предложения DEFAULT или оператора =. В качестве значения по умолчанию может быть использовано значение NULL, литерал и любая контекстная переменная совместимая по типу данных.

Обязательно используйте инициализацию начальным значением для любых переменных объявленных с ограничением NOT NULL, если они не получают значение по умолчанию иным способом.

Пример объявления локальной переменной

```
CREATE OR ALTER PROCEDURE SOME_PROC
        AS
        -- Объявление переменной типа INT
        DECLARE I INT;
        -- Объявление переменной типа INT не допускающей значение NULL
        DECLARE VARIABLE J INT NOT NULL;
        -- Объявление переменной типа INT со значением по умолчанию 0
        DECLARE VARIABLE K INT DEFAULT 0;
        -- Объявление переменной типа INT со значением по умолчанию 1
        DECLARE VARIABLE L INT = 1;
        -- Объявление переменной на основе домена COUNTRYNAME
        DECLARE FARM_COUNTRY COUNTRYNAME;
        -- Объявление переменной с типом равным типу домена COUNTRYNAME
        DECLARE FROM_COUNTRY TYPE OF COUNTRYNAME;
        -- Объявление переменной с типом столбца CAPITAL таблицы COUNTRY
        DECLARE CAPITAL TYPE OF COLUMN COUNTRY.CAPITAL;
BEGIN
        /* Операторы PSQL */
END
```

17.3.3 Объявление курсора

Для описания курсора также используется оператор DECLARE VARIABLE. Синтаксис оператора:

Листинг 17.2. Синтаксис оператора объявления локальной переменной и курсора DECLARE VARIABLE

```
DECLARE [VARIABLE] {
     <имя курсора> [SCROLL | NO SCROLL] CURSOR FOR (<оператор SELECT>)
}
```

Оператор DECLARE .. CURSOR FOR объявляет именованный курсор, связывая его с набором данных, полученным в операторе SELECT, указанном в предложении CURSOR FOR. В дальнейшем курсор может быть открыт, использоваться для обхода результирующего набора данных, и снова быть закрытым. Также поддерживаются позиционированные обновления и удаления при использовании WHERE CURRENT OF в операторах UPDATE и DELETE.

Имя курсора можно использовать в качестве ссылки на курсор, как на переменные типа запись. Текущая запись доступна через имя курсора, что делает необязательным предложение INTO в операторе FETCH.

Однонаправленные и прокручиваемые курсоры

Курсор может быть однонаправленными прокручиваемым. Необязательное предложение SCROLL делает курсор двунаправленным (прокручиваемым), предложение NO SCROLL - однонаправленным. По умолчанию курсоры являются однонаправленными.

Однонаправленные курсоры позволяют двигаться по набору данных только вперёд. Двунаправленные курсоры позволяют двигаться по набору данных не только вперёд, но и назад, а также на N позиций относительно текущего положения.

Прокручиваемые курсоры материализуются внутри как временный набор данных, таким образом, они потребляют дополнительные ресурсы памяти/диска, поэтому пользуйтесь ими только тогда, когда это действительно необходимо

Особенности использования курсора

Предложение FOR UPDATE разрешено использовать в операторе SELECT, но оно не требуется для успешного выполнения позиционированного обновления или удаления;

Удостоверьтесь, что объявленные имена курсоров не совпадают, ни с какими именами, определёнными позже в предложениях AS CURSOR;

Если курсор требуется только для прохода по результирующему набору данных, то практически всегда проще (и менее подвержено ошибкам) использовать оператор FOR SELECT с предложением AS CURSOR. Объявленные курсоры должны быть явно открыты, использованы для выборки данных и закрыты. Кроме того, вы должны проверить контекстную переменную ROW_COUNT после каждой выборки и выйти из цикла, если её значение ноль. Предложение FOR SELECT делает эту проверку автоматически. Однако объявленные курсоры дают большие возможности для контроля над последовательными событиями и позволяют управлять несколькими курсорами параллельно;

Оператор SELECT может содержать параметры, например: "SELECT NAME || :SFX FROM NAMES WHERE NUMBER = :NUM". Каждый параметр должен быть заранее объявлен как переменная PSQL (это касается также входных и выходных параметров). При открытии курсора параметру присваивается текущее значение переменной;

Если опция прокрутки опущена, то по умолчанию принимается NO SCROLL (т.е курсор открыт для движения только вперёд). Это означает, что могут быть использованы только команды FETCH [NEXT FROM]. Другие команды будут возвращать ошибки.

Если значение переменной PSQL, используемой в операторе SELECT, изменяется во время выполнения цикла, то её новое значение может (но не всегда) использоваться при выборке следующих строк. Лучше избегать таких ситуаций. Если вам действительно требуется такое поведение, то необходимо тщательно протестировать код и убедиться, что вы точно знаете, как изменения переменной влияют на результаты выборки. Поведение может зависеть от плана запроса, в частности, от используемых индексов. В настоящее время нет строгих правил для таких ситуаций, но в новых версиях это может измениться.

Пример использования именованного курсора

Объявление именованного курсора:

```
CREATE OR ALTER TRIGGER TBU_STOCK
BEFORE UPDATE ON STOCK
AS
        -- Объявление именованного курсора
        DECLARE C_COUNTRY CURSOR FOR (
                SELECT
                        COUNTRY,
                        CAPITAL
                FROM COUNTRY
        );
BEGIN
        /* Операторы PSQL */
END
```

Объявление прокручиваемого курсора:

```
EXECUTE BLOCK
RETURNS (
        N INT,
        RNAME CHAR(63))
AS
        -- Объявление прокручиваемого курсора
        DECLARE C SCROLL CURSOR FOR (
                SELECT
                        ROW_NUMBER() OVER(ORDER BY RDB$RELATION_NAME) AS N,
                        RDB$RELATION_NAME
                FROM RDB$RELATIONS
                ORDER BY RDB$RELATION_NAME);
BEGIN
        /* Операторы PSQL */
END
```

Следующий фрагмент хранимой процедуры выполняет выборку данных из таблицы стран. Результат полностью соответствует тому, который получается в случае выполнения оператора, описанного в другом примере.

```
DECLARE VARIABLE NEW_CURSOR
  CURSOR FOR (SELECT
              CODCOUNTRY, CODREGION, NAMEREG, CENTER
              FROM VIEW_RUSSIA2);
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
BEGIN
  OPEN NEW_CURSOR;
  WHILE (1 = 1) DO
  BEGIN
     FETCH NEW_CURSOR
         INTO :CODCOUNTRY, :CODREGION, :NAMEREG, :CENTER;
     IF (ROW_COUNT = 0) THEN
         LEAVE;
     SUSPEND;
  END
  CLOSE NEW_CURSOR;
END ^
```

17.4 Объявление и реализация подпроцедуры

Синтаксис объявления подпроцедуры:

```
<объявление подпроцедуры> ::=
 DECLARE PROCEDURE <имя подпроцедуры> [(<входной параметр> [, <входной> ...])]
 [RETURNS (<выходной параметр> [, <выходной параметр> ...])];

<реализация подпроцедуры> ::=
 DECLARE PROCEDURE <имя подпроцедуры> [(<входной> [, <входной> ...])]
 [RETURNS (<выходной параметр> [, <выходной параметр> ...])]
 <тело модуля>
```

Оператор `DECLARE PROCEDURE` объявляет или реализует подпроцедуру.

На подпроцедуру накладываются следующие ограничения:

- Подпрограмма не может быть вложена в другую подпрограмму. Они поддерживаются только в основном модуле (хранимой процедуре, хранимой функции, триггере и анонимном `PSQL` блоке);
- Переменные из основного модуля доступны внутри подпрограммы;
- При чтении переменные и параметры, к которым обращаются подпрограммы, могут иметь небольшое снижение производительности (даже в основной программе).
- В настоящее время подпрограмма не имеет прямого доступа до курсоров из основного модуля. Это может быть разрешено в будущем.

Одна подпрограмма может вызывать и другую подпрограмму, в том числе рекурсивно. В ряде случаев может потребоваться предварительное объявление подпрограммы. Общее правило: одна подпрограмма может вызвать другую подпрограмму, если последняя объявлена выше точки вызова. Все объявленные подпрограммы должны быть реализованы с той же сигнатурой. Значения по умолчанию для параметров подпрограмм не могут быть переопределены. Это означает, что они могут быть определены в реализации только тех подпрограмм, которые не были объявлены ранее.

Пример использования подпроцедуры:

```
SET TERM ^;
--
-- Подпроцедуры в EXECUTE BLOCK
--
EXECUTE BLOCK
RETURNS (
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
        name VARCHAR(63))
AS
        -- Подпроцедура, возвращающая список таблиц
        DECLARE PROCEDURE get_tables
        RETURNS(table_name VARCHAR(63))
        AS
                BEGIN
                        FOR
                                SELECT
                                        rdb$relation_name
                                FROM
                                        rdb$relations
                                WHERE
                                        rdb$view_blr IS NULL
                                INTO table_name
                        DO SUSPEND;
                END

-- Подпроцедура, возвращающая список представлений
DECLARE PROCEDURE get_views
RETURNS(view_name  VARCHAR(63))
AS
BEGIN
        FOR
                SELECT
                        rdb$relation_name
                FROM
                        rdb$relations
                WHERE
                        rdb$view_blr IS NOT NULL
                INTO view_name
        DO SUSPEND;
END
BEGIN
        FOR
                SELECT
                        table_name
                FROM
                        get_tables
                UNION ALL
                SELECT
                        view_name
                FROM
                        get_views
                INTO name
        DO SUSPEND;
END^
```

Пример использования подпроцедур с предварительным объявлением:

```
EXECUTE BLOCK RETURNS (o INTEGER)
AS
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
        -- Предварительное объявление P1.
        DECLARE PROCEDURE p1(i INTEGER = 1) RETURNS (o INTEGER);

        -- Предварительное объявление P2.
        DECLARE PROCEDURE p2(i INTEGER) RETURNS (o INTEGER);

        -- Реализация P1. Вы не должны переопределять значение параметра по умолчанию
        DECLARE PROCEDURE p1(i INTEGER) RETURNS (o INTEGER)
        AS
        BEGIN
                EXECUTE PROCEDURE p2(i) RETURNING_VALUES o;
        END

        DECLARE PROCEDURE p2(i INTEGER) RETURNS (o INTEGER)
        AS
        BEGIN
                o = i;
        END
BEGIN
        EXECUTE PROCEDURE p1 RETURNING_VALUES o;
SUSPEND;
END!
```

17.4.1 Объявление и реализация подфункции

Синтаксис объявления подфункции:

```
<объявление подфункции> ::=
 DECLARE FUNCTION <имя подфункции> [(<входной> [, <входной> ...])]
 RETURNS <тип> [COLLATE <сортировка>] [DETERMINISTIC];

<реализация подфункции> ::=
 DECLARE FUNCTION <имя подфункции> [(<входной> [, <входной> ...])]
 RETURNS <тип> [COLLATE <сортировка>] [DETERMINISTIC];
 <тело модуля>
```

Оператор `DECLARE FUNCTION` объявляет подфункцию.

На подфункцию накладываются следующие ограничения:

- Подпрограмма не может быть вложена в другую подпрограмму. Они поддерживаются только в основном модуле (хранимой процедуре, хранимой функции, триггере и анонимном `PSQL` блоке);
- Переменные из основного модуля доступны внутри подпрограммы;
- При чтении переменные и параметры, к которым обращаются подпрограммы, могут иметь небольшое снижение производительности (даже в основной программе).
- В настоящее время подпрограмма не имеет прямого доступа до курсоров из основного модуля. Это может быть разрешено в будущем.

Одна подпрограмма может вызывать и другую подпрограмму, в том числе рекурсивно. В ряде случаев может потребоваться предварительное объявление подпрограммы. Общее правило: одна подпрограмма может вызвать другую подпрограмму, если последняя объявлена выше точки вызова. Все объявленные подпрограммы должны быть реализованы с той же сигнатурой. Значения по умолчанию для параметров подпрограмм не могут быть переопределены. Это означает, что они могут быть

определены в реализации только тех подпрограмм, которые не были объявлены ранее.

17.4.2 Определение составного оператора

Синтаксис составного оператора:

```
<блок> ::=
BEGIN
        [<составной оператор> ...]
END
<составной оператор> ::= {<блок> | <оператор>}
```

Операторные скобки `BEGIN ... END` определяют составной оператор или блок операторов, который выполняется как одна единица кода. Каждый блок начинается оператором `BEGIN` и завершается оператором `END`. Блоки могут быть вложенными. Максимальная глубина ограничена 512 уровнями вложенности блоков. Составной оператор может быть пустым, что позволяет использовать его как заглушку, позволяющую избежать написания фиктивных операторов.

После операторов `BEGIN` и `END` точка с запятой не ставится. Однако утилита командной строки `isql` требует, чтобы после последнего оператора `END` в определении `PSQL` модуля следовал символ терминатора, установленного командой `SET TERM`. Терминатор не является частью синтаксиса `PSQL`.

Последний оператор `END` в триггере завершает работу триггера. Последний оператор `END` в хранимой процедуре работает в зависимости от типа процедуры:

- В селективной процедуре последний оператор `END` возвращает управление приложению и устанавливает значение `SQLCODE` равным 100, что означает, что больше нет строк для извлечения;
- В выполняемой процедуре последний оператор `END` возвращает управление и текущие значения выходных параметров, если таковые имеются, вызывающему приложению.

Для выполнения ветвления процесса обработки данных в `PSQL` используется оператор `IF-THEN-ELSE`. Его синтаксис представлен в листинге 17.3.

17.4.3 Оператор IF-THEN-ELSE

Оператор `IF-THEN-ELSE` выполняет условный переход.

Листинг 17.3. Синтаксис оператора `IF-THEN-ELSE`

```
IF (<условие>)
THEN <составной оператор>
[ELSE <составной оператор>];
```

- `Условие` - Логическое условие возвращающее `TRUE`, `FALSE` или `UNKNOWN`.
- `Составной оператор` - Оператор или блок операторов.

Оператор условного перехода `IF` используется для выполнения ветвления процесса обработки данных в `PSQL`. Если условие возвращает значение `TRUE`, то выполняется составной оператор или после ключевого слова `THEN`. Иначе (если условие возвращает `FALSE` или `UNKNOWN`) выполняется составной оператор после ключевого слова `ELSE`, если оно присутствует. Условие всегда заключается в круглые скобки.

`PSQL` не обеспечивает более сложных переходов с несколькими ветвями, таких как `CASE` или `SWITCH`. Однако можно объединить операторы `IF ... THEN ... ELSE` в цепочку. В качестве альтернативы, оператор `CASE` из `DSQL` доступен в `PSQL` и может удовлетворить по крайней мере некоторые варианты использования в виде `switch`:

```
CASE <условие>
        WHEN <выражение> THEN <результат>
        [WHEN <выражение> THEN <результат> ...]
        [ELSE <результат>]
END

CASE
        WHEN <логическое выражение> THEN <результат>
        [WHEN <логическое выражение> THEN <результат> ...]
        [ELSE <результат>]
END
```

```
...
C = CASE
        WHEN A=2 THEN 1
        WHEN A=1 THEN 3
        ELSE 0
END;
...
```

В следующем примере производится проверка на равенство пароля, введенного пользователем (INPUT_PASSWORD), паролю, прочитанному из таблицы базы данных (STORED_PASSWORD). В случае несоответствия паролей выдается пользовательское исключение с именем WRONG_PASSWORD, ранее созданное в базе данных.

```
IF (INPUT_PASSWORD <> STORED_PASSWORD)
THEN EXCEPTION WRONG_PASSWORD;
```

17.4.4 Оператор WHILE-DO

Оператор WHILE-DO позволяет организовать в PSQL цикл. Синтаксис оператора представлен в листинге 17.4.

Листинг 17.4. Синтаксис оператора WHILE-DO

```
[<метка>:]
WHILE (<условие>) DO
   <составной оператор>
```

- Условие - Логическое условие возвращающее TRUE, FALSE или UNKNOWN.
- Составной оператор - Оператор или блок операторов.

Оператор WHILE используется для организации циклов в PSQL. Составной оператор будет выполняться до тех пор, пока условие истинно (возвращает TRUE). Циклы могут быть вложенными, глубина вложения не ограничена.

В следующем фрагменте хранимой процедуры осуществляется расчет чисел Фибоначчи. Первые два числа (в приведенном фрагменте PREV_ITEM и NEXT_ITEM) имеют значение, соответственно, 1 и 2. Каждое следующее число в последовательности является суммой двух предыдущих. Вызвавшей программе в качестве выходного параметра RESULT возвращается последнее полученное число. Количество итераций задается входным целочисленным параметром LAST_NUM.

```
...
DECLARE VARIABLE I INTEGER;            -- Параметр цикла
DECLARE VARIABLE PREV_ITEM BIGINT;     -- Предыдущий элемент
DECLARE VARIABLE NEXT_ITEM BIGINT;     -- Следующий элемент
DECLARE VARIABLE INTERMEDIATE BIGINT;  -- Временный элемент
...
PREV_ITEM = 1;
NEXT_ITEM = 2;
I = 2;
INTERMEDIATE = NEXT_ITEM;
WHILE (I <= LAST_NUM) DO
BEGIN
   INTERMEDIATE = NEXT_ITEM;
   NEXT_ITEM = PREV_ITEM + NEXT_ITEM;
   PREV_ITEM = INTERMEDIATE;
   I = I + 1;
END
RESULT = NEXT_ITEM;
...
```

17.4.5 BREAK

Оператор BREAK осуществляет выход из цикла. Синтаксис оператора:

Листинг 17.5. Синтаксис оператора BREAK

```
<оператор цикла>
BEGIN
     ...
     BREAK;
     ...
     END

<оператор цикла> ::=
    FOR <оператор SELECT> INTO <список переменных> DO
  | FOR EXECUTE STATEMENT ... INTO <список переменных> DO
  | WHILE (<логическое условие>) DO
```

- Логическое условие - Логическое условие возвращающее TRUE, FALSE или UNKNOWN.

Оператор BREAK моментально прекращает работу внутреннего цикла операторов WHILE или FOR. Код продолжает выполняться с первого оператора после завершенного блока цикла. Оператор BREAK похож на LEAVE, за исключением того, что не поддерживает метку перехода.

17.4.6 LEAVE

Этот оператор осуществляет выход из цикла. Синтаксис оператора представлен в листинге 17.6.

Листинг 17.6. Синтаксис оператора LEAVE

```
[<метка>:]
<оператор цикла>
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
BEGIN
     ...
     LEAVE [метка];
     ...
     END

<оператор цикла> ::=
    FOR <оператор SELECT> INTO <список переменных> DO
  | FOR EXECUTE STATEMENT ... INTO <список переменных> DO
  | WHILE (<логическое условие>) DO
```

- `Логическое условие` - Логическое условие возвращающее `TRUE`, `FALSE` или `UNKNOWN`.

Оператор `LEAVE` немедленно прекращает работу внутреннего цикла операторов `WHILE` или `FOR`. С использованием необязательного параметра `<метка>`, `LEAVE` также может выйти и из внешнего цикла, то есть цикла помеченного меткой. Код продолжает выполняться с первого оператора после завершенного блока цикла.

Если в операторе не указана метка, то оператор просто осуществляет выход из текущего цикла `WHILE`. Пример расчета чисел Фибоначчи можно изменить, используя оператор `LEAVE`:

```
...
DECLARE VARIABLE I INTEGER;             -- Параметр цикла
DECLARE VARIABLE PREV_ITEM BIGINT;      -- Предыдущий элемент
DECLARE VARIABLE NEXT_ITEM BIGINT;      -- Следующий элемент
DECLARE VARIABLE INTERMEDIATE BIGINT;   -- Временный элемент
...
PREV_ITEM = 1;
NEXT_ITEM = 2;
I = 2;
INTERMEDIATE = NEXT_ITEM;
WHILE (1 = 1) DO
BEGIN
   IF (I > LAST_NUM) THEN
      LEAVE;
   INTERMEDIATE = NEXT_ITEM;
   NEXT_ITEM = PREV_ITEM + NEXT_ITEM;
   PREV_ITEM = INTERMEDIATE;
   I = I + 1;
END
RESULT = NEXT_ITEM;
...
```

Здесь в операторе `WHILE-DO` задается бесконечный цикл, поскольку условие выхода из цикла всегда истинно. Фактический выход из цикла осуществляется после соответствующей проверки условия в операторе `IF` с использованием оператора `LEAVE`.

17.4.7 CONTINUE

Оператор CONTINUE досрочно начинает новую итерацию цикла.

Синтаксис оператора:

Листинг 17.7. Синтаксис оператора CONTINUE

```
[<метка>:]
<оператор цикла>
BEGIN
     ...
     CONTINUE [метка];
     ...
END

<оператор цикла> ::=
    FOR <оператор SELECT> INTO <список переменных> DO
  | FOR EXECUTE STATEMENT ... INTO <список переменных> DO
  | WHILE (<логическое условие>) DO
```

- Логическое условие - Логическое условие возвращающее TRUE, FALSE или UNKNOWN.

Оператор CONTINUE пропускает оставшуюся часть текущего блока цикла и запускает следующую итерацию текущего цикла WHILE или FOR. С использованием необязательного параметра метка, CONTINUE также может начинать следующую итерацию для внешнего цикла, то есть цикла, помеченного меткой.

Использование оператора CONTINUE:

```
FOR
        SELECT A, D FROM ATABLE INTO :achar, :ddate
DO BEGIN
        IF (ddate < current_data - 30) THEN
                CONTINUE;
        ELSE
                /* do stuff */
        ...
END
```

17.4.8 EXIT

Оператор EXIT выполняет завершение работы процедуры, функции или триггера.

Листинг 17.8. Синтаксис оператора EXIT

```
EXIT;
```

Оператор EXIT, вызванный из любой точки выполняющегося PSQL модуля, переходит на последний оператор END, таким образом завершая выполнение программы. Вызов EXIT в функции приведет к тому, что функция вернет NULL.

17.4.9 SUSPEND

Оператор SUSPEND выполняет передачу значений параметров в буфер и приостановку выполнения процедуры (PSQL блока) до тех пор, пока вызывающая сторона не получит результат.

Листинг 17.9. Синтаксис оператора SUSPEND

```
SUSPEND;
```

Оператор SUSPEND передаёт значения выходных параметров в буфер и приостанавливает выполнение хранимой процедуры (PSQL блока). Выполнение остаётся приостановленным до тех пор, пока вызывающая сторона не получит содержимое буфера. Выполнение возобновляется с оператора, следующего непосредственно после оператора SUSPEND. Чаще всего это будет новой итерацией циклического процесса.

1. Оператор SUSPEND может встречаться только в хранимых процедурах или подпроцедурах, а также в анонимных блоках EXECUTE BLOCK.
2. Наличие ключевого слова SUSPEND определяет хранимую процедуру как выбираемую процедуру.
3. Приложения, использующие API интерфейсы, обычно делают выборку из хранимых процедур прозрачно.
4. Если выбираемая (selectable) процедура выполняется с использованием EXECUTE PROCEDURE, она ведет себя как исполняемая процедура. Когда в такой хранимой процедуре выполняется инструкция SUSPEND, это то же самое, что выполнение инструкции EXIT, что приводит к немедленному завершению процедуры.
5. Оператор SUSPEND "нарушает" атомарность блока, внутри которого он находится. В случае возникновения ошибки в селективной процедуре, операторы, выполненные после последнего оператора SUSPEND, будут откачены. Операторы, выполненные до последнего оператора SUSPEND, не будут откачены, если не будет выполнен откат транзакции.

17.4.10 EXECUTE STATEMENT

Оператор EXECUTE STATEMENT предназначен для выполнения динамически созданных SQL операторов. Синтаксис оператора представлен в листинге 17.10.

Листинг 17.10. Синтаксис оператора EXECUTE STATEMENT

```
<execute_statement> ::=
    EXECUTE STATEMENT <аргумент>
            [<опция> ...]
            [INTO [:] <внутренняя переменная> [, [:] <внутренняя переменная>...] ]

    <аргумент> ::=
      <непараметризованный SQL запрос>
    | (<параметризованный SQL запрос>) (<параметры>)

    <параметры> ::= <именованные параметры> | <позиционные параметры>

    <именованные параметры> ::=
    [EXCESS] <имя параметра>:=<выражение>
    [, [EXCESS] <имя параметра>:=<выражение> ...]

    <позиционные параметры> ::= <выражение> [, <выражение> ...]
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
<опция> ::=
  WITH {AUTONOMOUS | COMMON} TRANSACTION
| WITH CALLER PRIVILEGES
| AS USER <имя пользователя>
| PASSWORD <пароль>
| ROLE <роль>
| ON EXTERNAL [DATA SOURCE] <строка подключения>
```

- `Непараметризованный SQL запрос` - Строковый литерал или переменная, содержащая непараметризованный `SQL` запрос.
- `Параметризованный SQL запрос` - Строковый литерал или переменная, содержащая параметризованный `SQL` запрос.
- `Имя параметра` - Имя параметра `SQL` запроса.
- `Выражение` - Выражение для получения значения параметра запроса.
- `Имя пользователя` - Имя пользователя. Может быть строкой, `CURRENT_USER` или переменной.
- `Пароль` - Пароль. Может быть строкой или переменной.
- `Роль` - Роль. Может быть строкой, `CURRENT_ROLE` или переменной.
- `Строка подключения` - Строка соединения с удалённой БД Может быть строкой или переменной.

Оператор `EXECUTE STATEMENT` принимает строковый параметр и выполняет его, как будто это оператор `DSQL`. Если оператор возвращает данные, то с помощью предложения `INTO` их можно передать в локальные переменные.

Параметризованные операторы

В `DSQL` операторе можно использовать параметры. Параметры могут быть именованными и позиционными (безымянные). Значение должно быть присвоено каждому параметру.

Особенности использования параметризованных операторов:

1. Одновременное использование именованных и позиционных параметров в одном запросе запрещено;
2. Если у оператора есть параметры, они должны быть помещены в круглые скобки при вызове `EXECUTE STATEMENT`, независимо от вида их представления: непосредственно в виде строки, как имя переменной или как выражение;
3. Именованным параметрам должно предшествовать двоеточие (`:`) в самом операторе, но не при присвоении значения параметру;
4. Передача значений безымянным параметрам должна происходить в том же порядке, в каком они встречаются в тексте запроса;
5. Присвоение значений параметров должно осуществляться при помощи специального оператора `:=`, аналогичного оператору присваивания языка `Pascal`;
6. Каждый именованный параметр может использоваться в операторе несколько раз, но только один раз при присвоении значения;
7. Для позиционных параметров число подставляемых значений должно точно равняться числу параметров (вопросительных знаков) в операторе;
8. Необязательное ключевое слово `EXCESS` обозначает, что данный именованный параметр необязательно должен упоминаться в тексте запроса. Обратите внимание, что все не `EXCESS` параметры должны присутствовать в запросе.

WITH {AUTONOMOUS | COMMON} TRANSACTION

По умолчанию оператор выполняется в контексте текущей транзакции. При использовании предложения `WITH AUTONOMOUS TRANSACTION` запускается новая транзакция с такими же параметрами, как и текущая. Она будет подтверждена, если оператор выполнился без ошибок и отменена в противном случае. С предложением `WITH COMMON TRANSACTION` по возможности используется текущая транзакция.

Если оператор должен работать в отдельном соединении, то используется уже запущенная в этом соединении транзакция (если таковая транзакция имеется). В противном случае стартует новая транзакция с параметрами текущей транзакции. Любые новые транзакции, запущенные в режиме `COMMON`, подтверждаются или откатываются вместе с текущей транзакцией.

WITH CALLER PRIVILEGES

По умолчанию операторы `SQL` выполняются с правами текущего пользователя. Спецификация `WITH CALLER PRIVILEGES` добавляет к ним привилегии для вызова хранимой процедуры или триггера, так же как если бы оператор выполнялся непосредственно подпрограммой. `WITH CALLER PRIVILEGES` не имеет никакого эффекта, если также присутствует предложение `ON EXTERNAL`.

ON EXTERNAL [DATA SOURCE]

С предложением `ON EXTERNAL DATA SOURCE` оператор выполняется в отдельном соединении с той же или другой базой данных, возможно даже на другом сервере. Если строка подключения имеет значение `NULL` или `''` (пустая строка), предложение `ON EXTERNAL` считается отсутствующим и оператор выполняется для текущей базы данных.

AS USER, PASSWORD и ROLE

Необязательные предложения `AS USER`, `PASSWORD` и `ROLE` позволяют указывать от имени какого пользователя, и с какой ролью будет выполняться `SQL` оператор. То, как авторизуется пользователь и открыто ли отдельное соединение, зависит от присутствия и значений параметров `ON EXTERNAL [DATA SOURCE]`, `AS USER`, `PASSWORD` и `ROLE`.

При использовании предложения `ON EXTERNAL` открывается новое соединение и:

- Если присутствует, по крайней мере, один из параметров `AS USER`, `PASSWORD` и `ROLE`, то будет предпринята попытка нативной аутентификации с указанными значениями параметров (в зависимости от строки соединения - локально или удалённо). Для недостающих параметров не используются никаких значений по умолчанию;
- Если все три параметра отсутствуют, и строка подключения не содержит имени сервера (или `IP` адреса), то новое соединение устанавливается к локальному серверу с пользователем и ролью текущего соединения. Термин локальный означает компьютер, где установлен сервер. Это совсем не обязательно компьютер клиента;
- Если все три параметра отсутствуют, но строка подключения содержит имя сервера (или `IP` адреса), то будет предпринята попытка доверенной (`trusted`) авторизации к удалённому серверу. Если авторизация прошла, то удалённая операционная система назначит пользователю имя - обычно это учётная запись, под которой работает сервер РЕД Базы Данных.

Если предложение `ON EXTERNAL` отсутствует:

- Если присутствует, по крайней мере, один из параметров `AS USER`, `PASSWORD` и `ROLE`, то будет открыто соединение к текущей базе данных с указанными значениями параметров. Для недостающих параметров не используются никаких значений по умолчанию;
- Если все три параметра отсутствуют, то оператор выполняется в текущем соединении.

> Если значение параметра `NULL` или `''`, то весь параметр считается отсутствующим. Кроме того, если параметр считается отсутствующим, то `AS USER` прини-

мает значение CURRENT_USER, а ROLE - CURRENT_ROLE. Сравнение при авторизации сделано чувствительным к регистру: в большинстве случаев это означает, что имена пользователя и роли должны быть написаны в верхнем регистре.

Предостережения:

1. Не существует способа проверить синтаксис выполняемого SQL оператора;
2. Нет никаких проверок зависимостей для обнаружения удалённых столбцов в таблице или самой таблицы;
3. Выполнение оператора с помощью оператора EXECUTE STATEMENT значительно медленнее, чем при непосредственном выполнении;
4. Возвращаемые значения строго проверяются на тип данных во избежание непредсказуемых исключений преобразования типа. Например, строка '1234' преобразуется в целое число 1234, а строка 'abc' вызовет ошибку преобразования. В целом эта функция должна использоваться очень осторожно, а вышеупомянутые факторы всегда должны приниматься во внимание. Если такого же результата можно достичь с использованием PSQL и/или DSQL, то это всегда предпочтительнее.

17.4.11 Пул внешних подключений

Чтобы избежать задержек при частом использовании внешних соединений, подсистема внешних источников данных (EDS) использует пул внешних подключений. Пул сохраняет неиспользуемые внешние соединения в течении некоторого времени, что позволяет избежать затрат на подключение/отключение для часто используемых строк подключения.

Как работает пул соединений:

- каждое внешнее соединение связывается с пулом при создании;
- пул имеет два списка: неиспользуемых соединений и активных соединений;
- когда соединение становится неиспользуемым (т. е. у него нет активных запросов и нет активных транзакций), то оно сбрасывается и помещается в список ожидающих (при успешном завершении сброса) или закрывается (если при сбросе произошла ошибка). Соединение сбрасывается при помощи инструкции ALTER SESSION RESET;
- если пул достиг максимального размера, то самое старое бездействующее соединение закрывается;
- когда РЕД База Данных просит создать новое внешнее соединение, то пул сначала ищет кандидата в списке простаивающих соединений. Поиск основан на 4 параметрах:
  - строка подключения;
  - имя пользователя;
  - пароль;
  - роль.

  Поиск чувствителен к регистру;
- если подходящее соединение найдено, то проверятся живое ли оно;
- если соединение не прошло проверку, то оно удаляется и поиск повторяется (ошибка не возвращается пользователю);
- найденное (и живое) соединение перемещается из списка простаивающих соединение в список активных соединений и возвращается вызывающему;
- если имеется несколько подходящих соединений, то будет выбрано наиболее часто используемое;
- если нет подходящего соединения, то создаётся новое и помещается в список активных соединений;
- когда время жизни простаивающего соединения истекло, то оно удаляется из пула и закры-

вается.

Основные характеристики:

- отсутствие "вечных" внешних соединений;
- ограниченное количество неактивных (простаивающих) внешних соединений в пуле;
- поддерживает быстрый поиск среди соединений (по 4 параметрам указанным выше);
- пул является общим для всех внешних баз данных;
- пул является общим для всех локальных соединений, обрабатываемых данным процессом РЕД Базы Данных.

Параметры пула внешних соединений:

- время жизни соединения: временной интервал с момента последнего использования соединения, после истечении которого он будет принудительно закрыт. Параметр `ExtConnPoolLifeTime` в `firebird.conf`. По умолчанию равен 7200 секунд;
- размер пула: максимально допустимое количество незанятых соединений в пуле. Параметр `ExtConnPoolSize` в `firebird.conf`. По умолчанию равен 0, т.е. пул внешних соединений отключен.

Пулом внешних соединений, а также его параметрами можно управлять с помощью специальных операторов. Подробнее см. операторы `ALTER EXTERNAL CONNECTIONS POOL`.

Состояние пула внешних подключений можно запросить с использованием контекстных переменных в пространстве имен `SYSTEM`:

Таблица 17.2 — Переменные пространства имён `SYSTEM` для контроля пула внешних соединений

Имя переменной	Описание
`EXT_CONN_POOL_SIZE`	Размер пула
`EXT_CONN_POOL_LIFETIME`	Время жизни неактивных соединений
`EXT_CONN_POOL_IDLE_COUNT`	Текущее количество неактивных соединений в пуле
`EXT_CONN_POOL_ACTIVE_COUNT`	Текущее количество активных соединений в пуле

1. Внешние соединения используют по умолчанию предложение `WITH COMMON TRANSACTION` и остаются открытыми до закрытия текущей транзакции. Они могут быть снова использованы при последующих вызовах оператора `EXECUTE STATEMENT`, но только если строка подключения точно такая же. Если включен пул внешних соединений, то вместо закрытия соединения, такие соединения будут попадать в список неактивных (простаивающих) соединений;
2. Внешние соединения, созданные с использованием предложения `WITH AUTONOMOUS TRANSACTION`, закрываются после выполнения оператора или попадают в список неактивных соединений пула (если он включен);
3. Операторы `WITH AUTONOMOUS TRANSACTION` могут использовать соединения, которые ранее были открыты операторами `WITH COMMON TRANSACTION`. В этом случае использованное соединение остаётся открытым и после выполнения оператора, т.к. у этого соединения есть, по крайней мере, одна не закрытая транзакция. Если включен пул внешних соединений, то вместо закрытия соединения, такие соединения будут попадать в список неактивных (простаивающих) соединений.

1. При использовании предложения `WITH COMMON TRANSACTION` транзакции будут снова использованы как можно дольше. Они будут подтверждаться или откатываться вместе с текущей транзакцией;
2. При использовании предложения `WITH AUTONOMOUS TRANSACTION` всегда запускается новая транзакция. Она будет подтверждена или отменена сразу же после выполнения оператора;

При использовании предложения `ON EXTERNAL` дополнительное соединение всегда делается через так называемого внешнего провайдера, даже если это соединение к текущей базе данных. Одним из последствий этого является то, что вы не можете обработать исключение привычными способами. Каждое исключение, вызванное оператором, возвращает `eds_connection` или `eds_statement` ошибки. Для обработки исключений в коде `PSQL` вы должны использовать `WHEN GDSCODE eds_connection, WHEN GDSCODE eds_statement` или `WHEN ANY`.

Если предложение ON EXTERNAL не используется, то исключения перехватываются в обычном порядке, даже если это дополнительное соединение с текущей базой данных.

1. Набор символов, используемый для внешнего соединения, совпадает с используемым набором для текущего соединения.
2. Двухфазные транзакции не поддерживаются.

17.4.12 FOR SELECT-DO

Оператор FOR SELECT-DO является оператором цикла, выбирающим строки из таблицы, представления, хранимой процедуры выбора. Синтаксис оператора представлен в листинге 17.11.

Листинг 17.11. Синтаксис оператора цикла FOR SELECT-DO

```
[<метка>:]
FOR
   <оператор SELECT>
       INTO [:]<имя переменной/параметра> [, [:]<имя переменной/параметра> ...]
   [AS CURSOR <имя курсора>]
DO <составной оператор>;
```

- Метка - Необязательная метка для LEAVE и/или CONTINUE. Должна следовать правилам для идентикаторов.
- Имя курсора - Имя курсора. Должно быть уникальным среди имён переменных и курсоров PSQL модуля.
- Имя переменной - Имя локальной переменной или входного/выходного параметра.
- Составной оператор - Составной оператор (оператор или блок операторов).

Оператор FOR SELECT выбирает очередную строку из таблицы (представления, селективной хранимой процедуры), после чего выполняется составной оператор. В каждой итерации цикла значения полей текущей строки копируются в локальные переменные. Добавление предложения AS CURSOR делает возможным позиционное удаление и обновление данных. Операторы FOR SELECT могут быть вложенными.

Оператор FOR SELECT может содержать именованные параметры, которые должны быть предварительно объявлены в операторе DECLARE VARIABLE, или во входных (выходных) параметрах процедуры (PSQL блока).

Оператор FOR SELECT должен содержать предложение INTO, которое располагается в конце этого оператора, или предложение AS CURSOR. На каждой итерации цикла в список переменных указанных в предложении INTO копируются значения полей текущей строки запроса. Цикл повторяется, пока не будут прочитаны все строки. После этого происходит выход из цикла. Цикл также может быть завершён до прочтения всех строк при использовании оператора LEAVE.

Необъявленный курсор

Необязательное предложение AS CURSOR создаёт именованный курсор, на который можно ссылаться (с использованием предложения WHERE CURRENT OF) внутри составного оператора следующего после предложения DO, для того чтобы удалить или модифицировать текущую строку.

Разрешается использовать имя курсора как переменную типа запись (аналогично OLD и NEW в триггерах), что позволяет получить доступ к столбцам результирующего набора. Использование предложение AS CURSOR делает предложение INTO необязательным.

Правила для курсорных переменных:

- Для разрешения неоднозначности при доступе к переменной курсора перед именем курсора необходим префикс двоеточие;

- К переменной курсора можно получить доступ без префикса двоеточия, но в этом случае, в зависимости от области видимости контекстов, существующих в запросе, имя может разрешиться как контекст запроса вместо курсора;
- Переменные курсора доступны только для чтения;
- В операторе `FOR SELECT` без предложения `AS CURSOR` необходимо использовать предложение `INTO`. Если указано предложение `AS CURSOR`, предложение `INTO` не требуется, но разрешено;
- Чтение из переменной курсора возвращает текущие значения полей. Это означает, что оператор `UPDATE` (с предложением `WHERE CURRENT OF`) обновит также и значения полей в переменной курсора для последующих чтений. Выполнение оператора `DELETE` (с предложением `WHERE CURRENT OF`) установит `NULL` для значений полей переменной курсора для последующих чтений.
- Над курсором, объявленным с помощью предложения `AS CURSOR` нельзя выполнять операторы `OPEN`, `FETCH` и `CLOSE`;
- Убедитесь, что имя курсора, определённое здесь, не совпадает ни с какими именами, созданными ранее оператором `DECLARE VARIABLE`;
- Предложение `FOR UPDATE`, разрешённое для использования в операторе `SELECT`, не является обязательным для успешного выполнения позиционного обновления или удаления.

Пример

В главе 15 был показан пример создания представления `VIEW_RUSSIA2`, выбирающего все регионы страны Россия. Следующий пример показывает возможность использования этого представления в цикле `FOR SELECT-DO`.

```
FOR SELECT
   CODCOUNTRY, CODREGION, NAMEREG, CENTER
FROM VIEW_RUSSIA2
INTO
   :CODCOUNTRY, :CODREGION, :NAMEREG, :CENTER
DO
BEGIN
   EXECUTE PROCEDURE PROC_N (CODCOUNTRY, CODREGION, NAMEREG, CENTER)
   RETURNING_VALUES RESULT;
   SUSPEND;
END
```

Внутренние переменные `CODCOUNTRY, CODREGION, NAMEREG` и `CENTER` являются выходными параметрами этой хранимой процедуры.

После некоторой дополнительной обработки в хранимой процедуре `PROC_N` (эта процедура здесь не описана) очередной строки, полученной из представления, данная хранимая процедура приостанавливает свою работу (оператор `SUSPEND`) и передает управление вызвавшей программе.

17.4.13 FOR EXECUTE STATEMENT

Оператор предназначен для выполнения динамически созданных `SQL` операторов с возвратом нескольких строк данных.

Синтаксис оператора представлен в листинге 17.12.

Листинг 17.12. Синтаксис оператора цикла `FOR EXECUTE STATEMENT`

```
[метка:]
     FOR <определение EXECUTE STATEMENT> DO <составной оператор>
```

Оператор FOR EXECUTE STATEMENT используется (по аналогии с конструкцией FOR SELECT) для операторов SELECT или EXECUTE BLOCK, возвращающих более одной строки.

17.4.14 OPEN

Оператор открытия курсора.

Листинг 17.13. Синтаксис оператора открытия курсора OPEN

```
OPEN <имя курсора>;
```

Курсор с указанным именем должен быть предварительно объявлен с помощью оператора DECLARE CURSOR.

Оператор OPEN открывает ранее объявленный курсор, выполняет объявленный в нем оператор SELECT и получает записи из результирующего набора данных.

Если в операторе SELECT курсора имеются параметры, то они должны быть объявлены как локальные переменные или входные (выходные) параметры до того как объявлен курсор. При открытии курсора параметру присваивается текущее значение переменной.

17.4.15 FETCH

Данные очередной строки таблицы (представления) при использовании курсора получаются при выполнении оператора FETCH для этого курсора. Синтаксис оператора FETCH представлен в листинге 17.14.

Листинг 17.14. Синтаксис оператора чтения очередной строки из курсора FETCH

```
FETCH [<продвижение курсора> FROM] <имя курсора>
   [INTO [:]<внутр. переменная> [, [:]<внутр. переменная> ...]];

<продвижение курсора> ::=
  NEXT
| PRIOR
| FIRST
| LAST
| RELATIVE n
| ABSOLUTE n
```

Оператор FETCH выбирает следующую строку данных из результирующего набора данных курсора и присваивает значения столбцов в переменные PSQL. Оператор FETCH применим только к курсорам, объявленным в операторе DECLARE CURSOR.

Оператор FETCH может указывать в каком направлении и на сколько записей продвинется позиция курсора. Предложение NEXT допустимо использовать как с прокручиваемыми, там и не прокручиваемыми курсорами.

Остальные предложения допустимо использовать только с прокручиваемыми курсорами.

Опции прокручиваемого курсора:

- NEXT - перемещает указатель курсора на 1 запись вперёд. Это действие по умолчанию.
- PRIOR - перемещает указатель курсора на 1 запись назад.
- FIRST - перемещает указатель курсора на первую запись.
- LAST - перемещает указатель курсора на последнюю запись.
- ABSOLTE n - перемещает указатель курсора на указанную запись; n - целочисленное выражение, где 1 обозначает первую строку. Для отрицательных значений абсолютная позиция

берется с конца набора результатов, поэтому -1 указывает последнюю строку, -2 - предпоследнюю строку и т. д. Нулевое значение (0) будет располагаться перед первой строкой.

- `RELATIVE n` - перемещает курсор на n строк из текущей позиции; положительные числа перемещают указатель вперед, а отрицательные числа - назад; использование нуля (0) не приведет к перемещению курсора, а `ROW_COUNT` будет установлено в ноль, поскольку новая строка не была выбрана.

Необязательное предложение `INTO` помещает данные из текущей строки курсора в `PSQL` переменные.

Разрешается использовать имя курсора как переменную типа запись (аналогично `OLD` и `NEW` в триггерах), что позволяет получить доступ к столбцам результирующего набора.

Правила использования курсорных переменных:

- Для разрешения неоднозначности при доступе к переменной курсора перед именем курсора необходим префикс двоеточие;
- К переменной курсора можно получить доступ без префикса двоеточия, но в этом случае, в зависимости от области видимости контекстов, существующих в запросе, имя может разрешиться как контекст запроса вместо курсора;
- Переменные курсора доступны только для чтения;
- Чтение из переменной курсора возвращает текущие значения полей. Это означает, что оператор `UPDATE` (с предложением `WHERE CURRENT OF`) обновит также и значения полей переменной курсора для последующих чтений. Выполнение оператора `DELETE` (с предложением `WHERE CURRENT OF`) установит `NULL` для значений полей переменной курсора для последующих чтений.

Для проверки того, что записи набора данных исчерпаны, используется контекстная переменная `ROW_COUNT`, которая возвращает количество строк выбранных оператором. Если произошло чтение очередной записи из набора данных, то `ROW_COUNT` равняется единице, иначе нулю.

17.4.16 CLOSE

Оператор закрывает курсор и освобождает все ресурсы вычислительной системы, связанные с этим курсором и полученным набором данных. Синтаксис оператора приведен в листинге 17.15.

Листинг 17.15. Синтаксис оператора закрытия курсора `CLOSE`

```
CLOSE <имя курсора>;
```

Оператор `CLOSE` закрывает открытый курсор. Любые все ещё открытые курсоры будут автоматически закрыты после выполнения кода триггера, хранимой процедуры, функции или анонимного `PSQL` блока, в пределах кода которого он был открыт. Оператор `CLOSE` применим только к курсорам, объявленным в операторе `DECLARE CURSOR`.

17.4.17 IN AUTONOMOUS TRANSACTION

Оператор `IN AUTONOMOUS TRANSACTION` позволяет выполнить оператор или блок операторов в автономной транзакции. Синтаксис оператора представлен в листинге 17.16.

Листинг 17.16. Синтаксис оператора `IN AUTONOMOUS TRANSACTION`

```
IN AUTONOMOUS TRANSACTION DO <составной оператор>
```

Оператор `IN AUTONOMOUS TRANSACTION` позволяет выполнить составной оператор в автономной транзакции. Код, работающий в автономной транзакции, будет подтверждаться сразу же после успешного завершения независимо от состояния родительской транзакции. Это бывает нужно, когда опре-

делённые действия не должны быть отменены, даже в случае возникновения ошибки в родительской транзакции.

Автономная транзакция имеет тот же уровень изоляции, что и родительская транзакция. Любое исключение, вызванное или появившееся в блоке кода автономной транзакции, приведёт к откату автономной транзакции и отмене всех внесённых изменений. Если код будет выполнен успешно, то автономная транзакция будет подтверждена.

17.4.18 Отправка событий базы данных

Существует возможность отправки клиентским приложениям сообщений базы данных. Сообщением (событием, `event`) базы данных является произвольный текст. Такое событие может получить любое клиентское приложение, «прослушивающее» данное событие, то есть, сообщившее, что готово принять данный текст.

Часто события используются для того, чтобы проинформировать клиентские приложения, соединенные в настоящий момент с базой данных, о том, что были выполнены изменения в какой-либо таблице. Получив соответствующее сообщение, клиентская программа может, например, выполнить повторный запуск транзакции и переоткрыть набор данных, чтобы иметь возможность видеть новое состояние данных.

Для отправки сообщения (события) используется оператор `POST_EVENT`. Его синтаксис представлен в листинге 17.17.

Листинг 17.17. Синтаксис оператора передачи события `POST_EVENT`

```
POST_EVENT {<сообщение>};
```

`Сообщение` - это строка размером до 127 байт. Содержимое строки может быть строковым литералом, переменной или любым допустимым выражением SQL, которое возвращает строку.

Все действия по выборке, обработке и изменению данных в хранимых процедурах, хранимых функциях и триггерах выполняются в блоке операторов, который заключается в операторные скобки `BEGIN` и `END`.

Операторы в блоке выполняются последовательно. В PSQL существуют операторы ветвления (`IF-THEN-ELSE`), операторы цикла (`WHILE-DO`, `FOR SELECT-DO`, `FOR EXECUTE STATEMENT`), операторы обращения к данным в базе данных (операторы выборки данных, добавления, изменения, удаления).

Операторам цикла могут предшествовать метки — имя метки, после которой ставится двоеточие. Помеченные операторы цикла могут быть использованы в операторе `LEAVE`.

В любом месте текста, где допустим пробел, могут быть помещены комментарии, которые располагаются между символами `/*` и `*/`. Один такой комментарий может занимать любое количество строк. Существует и другая форма комментариев: подряд идущие два символа минус (`--`). Текст комментария в этом случае продолжается лишь до конца текущей строки.

17.4.19 RETURN

Возвращает значения из хранимой функции.

```
RETURN <значение>;
```

Выражением для возврата может быть любым выражением, совместимым с типом возвращаемого значения функции.

Оператор `RETURN` завершает выполнение функции и возвращает значение выражения. `RETURN` может использоваться только в `PSQL` функциях (хранимых и локальных функциях).

17.5 Обработка ошибок

В PSQL существуют операторы для обработки ошибок и исключений в модулях. Существует множество встроенных исключений, которые возникают в случае возникновения стандартных ошибок при работе с DML и DDL операторами.

17.5.1 Пользовательские исключения

В триггерах, хранимых процедурах и функциях (но не в клиентских приложениях) существует возможность выдачи пользовательских исключений (exception). В базе данных создается именованный объект данных, исключение, содержащий текст сообщения вызываемого исключения. Если в процессе выполнения триггера, хранимой процедуры или функции обнаруживается соответствующая ошибка в базе данных (обычно на содержательном, а не на формальном уровне), то можно вызвать пользовательское исключение.

Для вызова в триггере или в хранимой процедуре и функции пользовательского исключения нужно выполнить оператор EXCEPTION (листинг 17.18).

Листинг 17.18. Синтаксис оператора вызова пользовательского исключения EXCEPTION

```
EXCEPTION <имя пользовательского исключения> [<текст сообщения> | USING (<значение> [,
<значение>...])];
```

Оператор EXCEPTION возбуждает пользовательское исключение с указанным именем. При возбуждении исключения можно указать альтернативный текст сообщения, который будет возвращен вызывающему интерфейсу при возникновении исключения. Максимальная длина текстового сообщения составляет 1021 байт.

Текст сообщения исключения может содержать слоты для параметров, которые заполняются при возбуждении исключения. Для передачи значений параметров в исключение используется предложение USING. Параметры рассматриваются слева направо. Каждый параметр передаётся в оператор, возбуждающий исключение, как N-ый, N начинается с 1:

- Если N-ый параметр не передан, его слот не заменяется;
- Если передано значение NULL, слот будет заменён на строку '***null***';
- Если количество передаваемых параметров будет больше, чем содержится в сообщении исключения, то лишние будут проигнорированы;
- Максимальный номер параметра равен 9;
- Общая длина сообщения, включая значения параметров, ограничена 1053 байтами.

Пример.

В базе данных существует таблица PEOPLE. В ней для каждого человека можно указать код его матери и код его отца. Если при помещении в эту таблицу новой строки указать неправильный код, например, код матери, то можно, предварительно создав соответствующее пользовательское исключение, вызвать это исключение.

```
CREATE EXCEPTION NO_MOTHER
'В базе данных отсутствует запись, соответствующая матери человека по имени @1';
...
INSERT INTO PEOPLE (..., CODMOTHER, ...)
VALUES (..., 'DF', ...);
WHEN ANY
DO EXCEPTION NO_MOTHER USING(FULLNAME);
```

Здесь, если при добавлении новой записи человека в таблицу PEOPLE в базе данных в той же таблице PEOPLE будет отсутствовать строка, соответствующая матери вводимого человека, то программа (триггер, хранимая процедура или хранимая функция) в блоке обработки ошибок WHEN-DO выдаст пользовательское исключение с именем NO_MOTHER и текстом: 'В базе данных отсутствует запись, соответствующая матери человека'. Если в данной программе не задана обработка подобного исключения, то работа программы завершится. Новая запись не будет помещена в базу данных.

17.5.2 WHEN-DO

Для обработки ошибочных ситуаций базы данных и пользовательских исключений в языке хранимых процедур, функций и триггеров используется оператор WHEN-DO. Оператор позволяет перехватить любые указанные ошибки базы данных и/или пользовательские исключения (EXCEPTION) при обращении к базе данных. Синтаксис оператора представлен в листинге 17.19.

Листинг 17.19. Синтаксис оператора обработки ошибок базы данных или пользовательских исключений WHEN-DO

```
<блок операторов> ::=
 BEGIN
   [<составной оператор> ...]
   [<when_do> ...]
 END

<составной оператор> ::= {<блок операторов> | <оператор>}

<when_do>::=
 WHEN { <ошибка> [, <ошибка> ...] | ANY }
 DO <составной оператор>;

<ошибка> ::= {
   SQLCODE <код ошибки SQLCODE>
 | SQLSTATE <код ошибки SQLSTATE>
 | GDSCODE <название кода ошибки>
 | EXCEPTION <имя пользовательского исключения> }
```

Этот оператор должен находиться в самом конце блока, в котором происходят обращения к базе данных, которые могут вызвать ошибки базы данных, непосредственно перед последним оператором END.

В условии оператора до ключевого слова DO задается перечисление тех ситуаций, при которых будет выполняться составной оператор. Здесь можно через запятую перечислить произвольное количество значений кодов SQLCODE, GDSCODE, SQLSTATE, имен пользовательских исключений или задать ключевое слово ANY, которое означает, что обработка ошибочной ситуации будет выполняться при появлении любой ошибки базы данных и/или любого пользовательского исключения.

После ключевого слова DO помещается оператор или блок операторов, заключенных в операторные скобки BEGIN и END. В этом блоке выполняется обработка возникшей ситуации.

Оператор WHEN вызывается только тогда, когда произошло одно из указанных в его условии событий. В случае выполнения оператора (даже если в нем фактически не было выполнено никаких действий) ошибка или пользовательское исключение считается обработанным. При этом не прерываются и не отменяются действия триггера или хранимой процедуры или функции, где присутствует этот оператор, работа продолжается, как если бы никаких исключительных ситуаций не было.

Оператор перехватывает ошибки и исключения в текущем блоке операторов. Он также перехватывает подобные ситуации во вложенных блоках, если эти ситуации не были в них обработаны.

Следующий пример показывает вариант обработки ошибочной ситуации, возникающей при по-

пытке поместить в таблицу строку, имеющую дублирующее значение первичного или уникального ключа.

```
INSERT INTO COUNTRY (CODCOUNTRY) VALUES ('USA');
WHEN SQLCODE -803
DO ... ;
```

Однако подобное значение SQLCODE будет сформировано и при попытке добавления в базу данных новой строки, создающей дублирующее значение построенного пользователем индекса (см. Приложение Б). Чтобы более точно определить данную конкретную ошибочную ситуацию, следует использовать не код SQLCODE, а название кода GDSCODE.

```
INSERT INTO COUNTRY (CODCOUNTRY) VALUES ('USA');
WHEN GDSCODE unique_key_violation
DO ... ;
```

Этот обработчик срабатывает уже только на дублирование значения первичного ключа при помещении в новой записи в базу данных.

При возникновении ошибок базы данных или пользовательских исключений вначале отыскивается оператор WHEN-DO в текущем блоке. Если соответствующий оператор был найден, то выполняется обработка ситуации. Иначе происходит переход на блок операторов выше по иерархии вложенности операторов, пока не будет найден подходящий обработчик возникшей ситуации. Только в том случае, если не будет найден соответствующий оператор обработки данной ситуации, процедура, функция или триггер выдаст сообщение об ошибке.

Для перехвата и обработки пользовательского исключения используется оператор WHEN-DO с ключевым словом EXCEPTION. В следующем примере показан вариант выдачи пользовательского исключения NO_MOTHER. В том же самом блоке операторов или в любом вышележащем можно перехватить и обработать это пользовательское исключение, задав оператор:

```
WHEN EXCEPTION NO_MOTHER
DO ... ;
```

Коды ошибок SQLCODE и GDSCODE детально описаны в Приложение Б.

Глава 18

Хранимые процедуры (PROCEDURE)

Хранимая процедура, так же как и триггер является программой, хранящейся в области метаданных базы данных и выполняющейся на стороне сервера. В отличие от триггера к хранимой процедуре могут обращаться хранимые процедуры, триггеры и клиентские программы. Допустима рекурсия — хранимая процедура может обращаться сама к себе. Хранимые процедуры выполняются в контексте той же транзакции, что и вызывающие их программы.

Преимущества хранимых процедур:

- Модульность - приложения, работающие с одной и той же базой данных, могут использовать одну и ту же хранимую процедуру, тем самым уменьшив размер кода приложения и устранив дублирование кода;
- Упрощение поддержки приложений - при изменении хранимой процедуры, изменения отражаются сразу во всех приложениях, перекомпиляции;
- Увеличение производительности - поскольку хранимые процедуры выполняются на стороне сервера, а не клиента, то это уменьшает сетевой трафик, что повышает производительностьь.

Существует два вида хранимых процедур — выполняемые хранимые процедуры (`executable stored procedures`) и селективные процедуры (`selectable stored procedures`).

Выполняемые хранимые процедуры, осуществляют обработку данных, находящихся в базе данных. Эти процедуры могут получать входные параметры и возвращать одиночный набор выходных (`RETURNS`) параметров. Такие процедуры выполняются с помощью оператора `EXECUTE PROCEDURE`.

Селективные хранимые процедуры обычно осуществляют выборку данных из базы данных и возвращают при этом произвольное количество строк.

Такие процедуры позволяют получать довольно сложные наборы данных, которые зачастую невозможно или весьма затруднительно получить с помощью обычных `DSQL SELECT` запросов. Обычно такие процедуры выполняют циклический процесс извлечения данных, возможно преобразуя их, прежде чем заполнить выходные переменные (параметры) новыми данными на каждой итерации цикла. Оператор `SUSPEND`, обычно расположенный в конце каждой итерации, заполняет буфер и ожидает пока вызывающая сторона не выберет (`fetch`) строку.

Селективные процедуры могут иметь входные параметры и выходное множество, заданное в предложении `RETURNS` в заголовке процедуры.

Обращение к селективной хранимой процедуре осуществляется при помощи оператора `SELECT`.

18.1 Создание хранимой процедуры

Синтаксис создания выполняемых хранимых процедур и селективных процедур ничем не отличается. Разница заключается в логике программного кода. Для создания хранимой процедуры используется оператор `CREATE PROCEDURE`, синтаксис которого представлен в листинге 18.1.

Листинг 18.1. Синтаксис оператора создания хранимой процедуры `CREATE PROCEDURE`

```
CREATE PROCEDURE <имя хранимой процедуры>
[AUTHID {OWNER | CALLER}]
   [(<входной параметр> [, <входной параметр> ...])]
[RETURNS (<выходной параметр> [, <выходной параметр> ...])]
[SQL SECURITY {DEFINER | INVOKER}]
{ EXTERNAL NAME '<внешний модуль>' ENGINE <имя движка> [AS <тело внешней проц.>] }
|
{
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
    {AS '<BLR код>'}
    |
    {AS
      [<объявление> [<объявление> ...] ]
    BEGIN
      <блок операторов>
    END }
}

<входной параметр> ::= <описание параметра> [{=|DEFAULT} <значение по умолчанию>]

<выходной параметр> ::= <описание параметра>

<описание параметра> ::= <имя параметра> <тип> [NOT NULL] [COLLATE <порядок
сортировки>]

<тип> ::= { <тип данных SQL>
          | [TYPE OF] <имя домена>
          | TYPE OF COLUMN <имя таблицы/представления>.<имя столбца> }

<значение по умолчанию>::= {<литерал> | NULL | <контекстная переменная>}

<внешний модуль> ::= '<имя внешнего модуля>!<имя функции в модуле>[! <информация>]'

<объявление> ::= <объявление локальной переменной>;
               | <объявление курсора>;
               | <объявление процедуры/функции>
               | <реализация процедуры/функции>
```

Таблица 18.1 — Параметры оператора CREATE PROCEDURE

Параметр	Описание
Имя хранимой процедуры	Имя хранимой процедуры. Может содержать до 63 символов.
Входной параметр	Описание входного параметра.
Выходной параметр	Описание выходного параметра.
Литерал	Литерал, совместимый по типу с параметром.
Контекстная переменная	Любая контекстная переменная, тип которой совместим с типом параметра.
Имя параметра	Имя входного или выходного параметра процедуры. Может содержать до 63 символов. Имя параметра должно быть уникальным среди входных и выходных параметров процедуры, а также её локальных переменных.
Тело внешней процедуры	Тело внешней процедуры. Строковый литерал который может использоваться UDR для различных целей.
Имя внешнего модуля	Имя внешнего модуля, в котором расположена функция.
Имя функции в модуле	Внутреннее имя функции внутри внешнего модуля.
Информация	Определяемая пользователем информация для передачи в функцию внешнего модуля.

(разрыв таблицы)

(разрыв таблицы)

Параметр	Описание
`Имя движка`	Имя движка для использования внешних функций. Обычно указывается имя UDR.

Хранимую процедуру может создать администратор и пользователь с привилегией `CREATE PROCEDURE`. Пользователь, создавший хранимую процедуру, становится её владельцем.

Оператор `CREATE PROCEDURE` создаёт новую хранимую процедуру. Имя хранимой процедуры должно быть уникальным среди имён всех хранимых процедур, таблиц и представлений базы данных.

Желательно также, чтобы имя хранимой процедуры было уникальным и среди имён процедур расположенных в PSQL пакетах, хотя это и допустимо. Дело в том, что в настоящее время вы не сможете вызвать функцию/процедуру из глобального пространства имён внутри пакета, если в пакете объявлена одноименная функция/процедура. В этом случае всегда будет вызвана процедура/функция пакета.

`CREATE PROCEDURE` является составным оператором, состоящим из заголовка и тела.

Заголовок определяет имя хранимой процедуры и объявляет входные и выходные параметры, если они должны быть возвращены процедурой.

Тело процедуры состоит из необязательных объявлений локальных переменных, подпрограмм и именованных курсоров, и одного или нескольких операторов, или блоков операторов, заключённых во внешнем блоке, который начинается с ключевого слова `BEGIN`, и завершается ключевым словом `END`. Объявления локальных переменных и именованных курсоров, а также внутренние операторы должны завершаться точкой с запятой (`;`).

18.1.1 Привилегии выполнения

Чтобы указать в контексте какого пользователя будет выполняться процедура используются необязательные предложения `AUTHID` или `SQL SECURITY`. Совместное их использование недопустимо. Причем предложение `AUTHID` считается устаревшим и не будет поддерживаться начиная с РЕД Базы Данных 6.

Используйте следующие предложения, чтобы процедура выполнялась:

- с правами вызывающего ее пользователя (значение по умолчанию):

```
CREATE PROCEDURE MyProc AUTHID CALLER (...)
RETURNS (...)
AS BEGIN
...
END!
```

или

```
CREATE PROCEDURE MyProc (...)
RETURNS (...)
SQL SECURITY INVOKER
AS BEGIN
...
END!
```

- с правами ее владельца (создателя):

```
CREATE PROCEDURE MyProc AUTHID OWNER (...)
RETURNS (...)
AS BEGIN
...
END!
```

```
CREATE PROCEDURE MyProc (...)
RETURNS (...)
SQL SECURITY DEFINER
AS BEGIN
...
END!
```

Значение по умолчанию на уровне всей базы данных можно изменить оператором `ALTER DATABASE SET DEFAULT SQL SECURITY`.

18.1.2 Параметры

У каждого параметра указывается тип данных. Кроме того, для параметра можно указать ограничение `NOT NULL`, тем самым запретив передавать в него значение `NULL`.

Для параметра строкового типа существует возможность задать порядок сортировки с помощью предложения `COLLATE`.

Входные параметры

Входные параметры заключаются в скобки после имени хранимой процедуры. Они передаются в процедуру по значению, то есть любые изменения входных параметров внутри процедуры никак не повлияет на значения этих параметров в вызывающей программе.

Входные параметры могут иметь значение по умолчанию. Параметры, для которых заданы значения, должны располагаться в конце списка параметров.

Выходные параметры

Необязательное предложение `RETURNS` позволяет задать список выходных параметров хранимой процедуры.

Использование доменов при объявлении параметров

Если при описании параметра, локальной переменной процедуры указано имя домена, то для него копируются все характеристики этого домена. Если в описании присутствует предложение `TYPE OF`, то для переменной копируется только тип данных домена, а в случае строковых типов ещё и набор символов, и порядок сортировки.

Использование доменов при объявлении параметров

В качестве типа параметра можно указать имя домена. В этом случае параметр будет наследовать все характеристики домена.

Если перед названием домена дополнительно используется предложение `TYPE OF`, то используется только тип данных домена - не проверяются его ограничения `NOT NULL` и `CHECK` (если они есть) и не используется значение по умолчанию. Если домен текстового типа, то всегда используется его набор символов и порядок сортировки.

Использование типа столбца при объявлении параметров

Входные и выходные параметры можно объявлять, используя тип данных столбцов существующих таблиц и представлений. Для этого используется предложение `TYPE OF COLUMN`, после которого указывается имя таблицы или представления и через точку имя столбца.

При использовании `TYPE OF COLUMN` наследуется только тип данных, а в случае строковых типов ещё и набор символов, и порядок сортировки. Ограничения и значения по умолчанию столбца никогда не используются.

18.1.3 Тело хранимой процедуры

После ключевого слова `AS` следует тело хранимой процедуры.

Объявление локальных переменных, курсоров и подпрограмм

В необязательной секции `<объявление>` описаны локальные переменные процедуры, подпрограммы и именованные курсоры. В отношении спецификации типа данных локальные переменные подчиняются тем же правилам, что и входные и выходные параметры процедуры.

После необязательной секции деклараций обязательно следует составной оператор. Составной оператор состоит из одного или нескольких `PSQL` операторов, заключенных между ключевыми словами `BEGIN` и `END`. Составной оператор может содержать один или несколько других составных операторов. Вложенность ограничена 512 уровнями. Любой из `BEGIN ... END` блоков может быть пустым, в том числе и главный блок.

18.1.4 Внешние хранимые процедуры

Хранимая процедура может быть расположена во внешнем модуле. В этом случае вместо тела процедуры указывается место её расположения во внешнем модуле с помощью предложения `EXTERNAL NAME`. Аргументом этого предложения является строка, в которой через разделитель указано имя внешнего модуля, имя процедуры внутри модуля и определённая пользователем информация.

В предложении `ENGINE` указывается имя движка для обработки подключения внешних модулей. В РЕД Базе Данных для работы с внешними модулями используется движок `UDR`. После ключевого слова `AS` может быть указан строковый литерал - тело внешней процедуры, оно может быть использовано внешним модулем для различных целей.

18.1.5 Привилегии выполнения

Чтобы указать в контексте какого пользователя будет выполняться процедура используется необязательное предложение `SQL SECURITY`. Если выбрана опция `INVOKER`, то хранимая процедура выполняется с привилегиями вызывающего пользователя. Если выбрана опция `DEFINER`, то хранимая процедура выполняется с привилегиями определяющего пользователя (владельца процедуры). Эти привилегии будут дополнены привилегиями выданные самой хранимой процедуре с помощью оператора `GRANT`. По умолчанию хранимая процедура выполняется с привилегиями вызывающего пользователя

Значение по умолчанию на уровне всей базы данных можно изменить оператором `ALTER DATABASE SET DEFAULT SQL SECURITY`.

18.1.6 Использование BLR вместо SQL кода

В РЕД Базе Данных 5.0 появилась возможность создать процедуру с телом в виде двоичного BLR представления, закодированного в `Base64`, вместо обычного исходного кода на языке `SQL`. В этом случае исходные `SQL` коды процедуры будут недоступны. Оператор создания такой процедуры будет выглядеть так:

```
CREATE PROCEDURE <имя> (<входные параметры>)
RETURNS (<выходные параметры>)
[SQL SECURITY {DEFINER | INVOKER}]
AS '<BLR код>';
```

Здесь <BLR код> - это BLR в его исходном двоичном виде, закодированный в Base64. Его можно получить, например, из запроса:

```
select BASE64_ENCODE(RDB$PROCEDURE_BLR)
from RDB$PROCEDURES
where RDB$PROCEDURE_NAME = '<имя>';
```

18.2 Изменение хранимой процедуры

Для изменения существующей хранимой процедуры используется оператор ALTER PROCEDURE. Синтаксис оператора представлен в листинге 18.2.

Листинг 18.2. Синтаксис оператора изменения хранимой процедуры ALTER PROCEDURE

```
ALTER PROCEDURE <имя хранимой процедуры>
[AUTHID {OWNER | CALLER}]
   [(<входной пар-р>[,<входной пар-р>...])]
[RETURNS (<выходной параметр> [, <выходной параметр> ...])]
[SQL SECURITY {DEFINER | INVOKER}]
{ EXTERNAL NAME '<внешний модуль>' ENGINE <имя движка> [AS <тело внешней проц.>]} |
{
   {AS '<BLR код>'}
   |
   {AS
      [<объявление> [<объявление> ...] ]
   BEGIN
      <блок операторов>
   END }
}
```

Оператор позволяет изменять:

- состав и характеристики входных параметров;
- состав и характеристики выходных параметров;
- в контексте какого пользователя будет выполняться процедура;
- список локальных переменных и именованных курсоров;
- тело хранимой процедуры на языке SQL;
- исходный SQL код процедуры на двоичный BLR код;
- для внешних процедур (UDR) — точку входа и имя движка.

В одном операторе ALTER PROCEDURE можно изменять любую из перечисленных частей или все сразу.

Выполняемые изменения в хранимой процедуре не оказывают никакого влияния на ее зависимости и привилегии.

Изменять хранимую процедуру может ее создатель и администратор (пользователь с ролью RDB$ADMIN) и пользователь с привилегией ALTER ANY PROCEDURE.

Будьте осторожны при изменении количества и типов входных и выходных параметров хранимых процедур. Существующий код приложения может стать неработоспособным из-за того, что формат вызова процедуры несовместим с новым описанием параметров. Кроме того, PSQL модули, использующие изменённую хранимую процедуру, могут стать некорректными. Информация о том, как это обнаружить, находится в разделе Приложение B.

Изменить значение SQL SECURITY можно без указания тела процедуры:

```
ALTER PROCEDURE <имя хранимой процедуры>
  SQL SECURITY {DEFINER | INVOKER}
| DROP SQL SECURITY
```

18.3 Создание новой или изменение существующей хранимой процедуры

Оператор CREATE OR ALTER PROCEDURE создаёт новую или изменяет существующую хранимую процедуру. Если хранимая процедура не существует, то она будет создана с использованием предложения CREATE PROCEDURE. Если она уже существует, то она будет изменена и откомпилирована, при этом существующие привилегии и зависимости сохраняются. Синтаксис оператора представлен в листинге 18.3.

Листинг 18.3. Синтаксис оператора создания новой или изменения существующей хранимой процедуры CREATE OR ALTER PROCEDURE

```
CREATE OR ALTER PROCEDURE <имя хранимой процедуры>
[AUTHID {OWNER | CALLER}]
   [(<входной параметр> [, <входной параметр> ...])]
[RETURNS (<выходной параметр> [, <выходной параметр> ...])]
[SQL SECURITY {DEFINER | INVOKER}]
{ EXTERNAL NAME '<внешний модуль>' ENGINE <имя движка> [AS <тело внешней проц.>]} |
{
   {AS '<BLR код>'}
   |
   {AS
      [<объявление> [<объявление> ...] ]
   BEGIN
      <блок операторов>
   END }
}
```

Семантика операторов и предложений в этом операторе полностью соответствует оператору CREATE PROCEDURE.

18.4 Удаление хранимой процедуры

Для удаления существующей хранимой процедуры используется оператор DROP PROCEDURE. Синтаксис оператора представлен в листинге 18.4.

Листинг 18.4. Синтаксис оператора удаления хранимой процедуры DROP PROCEDURE

```
DROP PROCEDURE <имя хранимой процедуры>
```

Если от хранимой процедуры существуют зависимости, то при попытке удаления такой процедуру будет выдана соответствующая ошибка. Также нельзя удалить хранимую процедуру, которая выполняется в настоящий момент.

Удалить хранимую процедуру может ее создатель и администратор (пользователь с ролью RDB$ADMIN) и пользователь с привилегией DROP ANY PROCEDURE.

18.5 Создание новой или пересоздание существующей хранимой процедуры

Оператор RECREATE PROCEDURE создает новую или пересоздает существующую хранимую процедуру. Если процедура с таким именем уже существует, то оператор попытается удалить ее и создать новую процедуру, при этом привилегии на выполнение хранимой процедуры и привилегии самой хранимой процедуры не сохраняются.

Листинг 18.5. Синтаксис оператора пересоздания хранимой процедуры RECREATE PROCEDURE

```
RECREATE PROCEDURE <имя хранимой процедуры>
[AUTHID {OWNER | CALLER}]
   [(<входной параметр> [, <входной параметр> ...])]
[RETURNS (<выходной параметр> [, <выходной параметр> ...])]
[SQL SECURITY {DEFINER | INVOKER}]
{ EXTERNAL NAME '<внешний модуль>' ENGINE <имя движка> [AS <тело внешней проц.>]} |
{
   {AS '<BLR код>'}
   |
   {AS
      [<объявление> [<объявление> ...] ]
   BEGIN
      <блок операторов>
   END }
}
```

Операция закончится неудачей при подтверждении транзакции, если процедура имеет зависимости. Ошибки зависимостей не обнаруживаются до фазы подтверждения транзакции.

18.6 Примеры хранимых процедур

Преимуществами хранимых процедур является то, что они, во-первых, выполняются на стороне сервера, что во многих случаях может резко сократить сетевой трафик, во-вторых, один раз написан-

ная и отлаженная хранимая процедура может использоваться многими программами — хранимыми процедурами, триггерами, клиентскими программами.

Выполняемые хранимые процедуры могут получать входные параметры и возвращать вызвавшей программе выходные параметры. Обращение к выполняемой хранимой процедуре осуществляется при помощи оператора `EXECUTE PROCEDURE`.

Селективная хранимая процедура используется, как правило, для выборки достаточно большого количества данных из базы данных. Алгоритм выборки данных в таких случаях достаточно сложный. Подобного вида процедуры обычно используются в том случае, когда декларативных средств оператора `SELECT` недостаточно для выполнения всех действий по выборке релевантных данных из таблиц или представлений. Такая процедура также может получать входные параметры и возвращать выходные параметры.

Далее в этой главе рассматриваются простые примеры хранимых процедур.

18.6.1 Получение значения искусственного первичного ключа

Для получения значения искусственного первичного ключа из генератора клиентской программой для таблицы `PEOPLE` можно использовать выполняемую хранимую процедуру:

```
SET TERM ^;
CREATE PROCEDURE PROC_PEOPLE
RETURNS (COD INTEGER)
AS
BEGIN
   COD = NEXT VALUE FOR GEN_PEOPLE;
   SUSPEND;
END ^
```

Сравните этот текст с примером, где описано создание триггера, выполняющего эти же функции. Использование триггера для получения значения искусственного ключа имеет то единственное преимущество, что триггер вызывается автоматически перед помещением новой строки в таблицу. От разработчика клиентской программы не требуется никаких специальных действий для этого. Однако в случае использования триггера клиентская программа не знает полученного числового значения. В некоторых ситуациях это может затруднить дальнейшую работу программы.

При использовании хранимой процедуры для получения значения первичного ключа клиентская программа явно получает соответствующее значение и использует его в операторе `INSERT` и в последующих операциях.

18.6.2 Вычисление факториала числа

В главе 11 был приведен пример оператора `EXECUTE BLOCK`, который позволял вычислить факториал заданного числа в декларативной части SQL.

В следующем примере приведены операторы создания хранимой процедуры, выполняющей вычисление факториала целого числа. В процедуре также присутствует оператор `WHEN-DO`, который обрабатывает ошибочную ситуацию — арифметическое переполнение, когда результат превышает значение, которое может быть помещено в переменную с типом данных `BIGINT`.

```
CREATE OR ALTER PROCEDURE PROC_FACTORIAL (N BIGINT)
RETURNS (RESULT BIGINT, TEXT VARCHAR(50))
AS
   DECLARE VARIABLE I BIGINT;
BEGIN
   RESULT = 1; I = 1;
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
    TEXT = 'That''s OK';
    WHILE (I <= N) DO
    BEGIN
       RESULT = RESULT * I; I = I + 1;
       WHEN ANY DO
       BEGIN
          TEXT = 'Overflow';
          RESULT = NULL;
          LEAVE;
       END
    END
    SUSPEND;
END ^
```

Здесь в цикле WHILE-DO осуществляются необходимые вычисления. Если не произошло арифметического переполнения, процедура возвращает в первом выходном параметре полученное число, а во втором символьном параметре текст "That's OK". В случае переполнения ошибочную ситуацию перехватывает оператор WHEN-DO. В нем формируется пустое значение результата, в символьный выходной параметр помещается текст "Overflow" и осуществляется выход из цикла при помощи оператора LEAVE.

Для того чтобы процедуру можно было вызвать при помощи оператора SELECT и отобразить полученный результат, в текст процедуры введен оператор SUSPEND. Вызвать эту процедуру можно, используя, например, следующий оператор SELECT:

```
SELECT * FROM PROC_FACTORIAL (20);
```

Результатом будет число 2432902008176640000 и текст "That's OK".

В этом примере вместо оператора CREATE PROCEDURE используется оператор CREATE OR ALTER PROCEDURE. Если процедура с таким именем уже существует в базе данных, то она будет заменена на новую с тем же именем без выдачи диагностических сообщений.

В следующем примере создается рекурсивная хранимая процедура, вычисляющая факториал.

```
CREATE PROCEDURE RECFACTORIAL (NUM INTEGER)
RETURNS (N_FACTORIAL DOUBLE PRECISION)
AS
   DECLARE VARIABLE NUM_LESS_ONE INT;
BEGIN
   IF (NUM = 1) THEN BEGIN        /**** Простейший случай: 1! = 1 ****/
      N_FACTORIAL = 1;
      SUSPEND;
   END
   ELSE BEGIN                     /**** Рекурсия: NUM! = (NUM * (NUM-1))! ****/
      NUM_LESS_ONE = NUM - 1;
      EXECUTE PROCEDURE RECFACTORIAL (NUM_LESS_ONE)
      RETURNING_VALUES N_FACTORIAL;
      N_FACTORIAL = N_FACTORIAL * NUM;
      SUSPEND;
   END
END ^
```

Количество возможных рекурсий устанавливается не более 1000 во избежание зацикливания. В реальности, в зависимости от доступных ресурсов серверной машины, это число может быть меньшим.

В последнем примере в качестве типа данных для выходного параметра был выбран тип данных DOUBLE PRECISION. Это является более разумным решением в том случае, если получаемое значение превышает максимально допустимое для типа данных BIGINT. В первом примере вычисления факториала выбран другой тип данных только для того, чтобы проиллюстрировать средства обработки ошибок в PSQL. Это касается и многих других примеров данной главы.

В главе 17 был приведен пример использования операторов PSQL для получения чисел Фибоначчи. В следующем примере задается полный текст реальной хранимой процедуры, вычисляющей последнее значение числа Фибоначчи в заданном ряду. Первые два элемента в ряду 1 и 2. Каждый последующий элемент является суммой двух предшествующих.

```
CREATE PROCEDURE FIBONACCI_LAST (LAST_NUM INT)
RETURNS (RESULT BIGINT)
AS
   DECLARE VARIABLE I INTEGER;              -- Параметр цикла
   DECLARE VARIABLE PREV_ITEM BIGINT;       -- Предыдущий элемент
   DECLARE VARIABLE NEXT_ITEM BIGINT;       -- Следующий элемент
   DECLARE VARIABLE INTERMEDIATE BIGINT;    -- Временный элемент
BEGIN
   PREV_ITEM = 1;
   NEXT_ITEM = 2;
   I = 2;
   INTERMEDIATE = NEXT_ITEM;
   WHILE (I <= LAST_NUM) DO BEGIN
      INTERMEDIATE = NEXT_ITEM;
      NEXT_ITEM = PREV_ITEM + NEXT_ITEM;
      PREV_ITEM = INTERMEDIATE;
      I = I + 1;
   END
   RESULT = NEXT_ITEM;
   SUSPEND;
END ^
```

Это выполняемая хранимая процедура, которая выполняет вычисления и возвращает вызвавшей программе ровно одно значение. Оператор SUSPEND здесь присутствует лишь для того, чтобы результат можно было отобразить при обращении к процедуре с помощью оператора SELECT.

В следующем примере приведен текст еще одной селективной хранимой процедуры для вычисления чисел Фибоначчи. В отличие от предыдущей процедуры в данном случае выводится не только последнее число, но и все числа полученного ряда. Рассчитывается также частное от деления последующего элемента списка на предыдущий элемент.

```
CREATE PROCEDURE FIBONACCI_SEQ (LAST_NUM INT)
RETURNS (RESULT BIGINT, QUOTIENT DOUBLE PRECISION)
AS
   DECLARE VARIABLE I INTEGER;              -- Параметр цикла
   DECLARE VARIABLE PREV_ITEM BIGINT;       -- Предыдущий элемент
   DECLARE VARIABLE INTERMEDIATE BIGINT;    -- Временный элемент
BEGIN
   RESULT = 1;
   QUOTIENT = 1;
   SUSPEND;
   PREV_ITEM = 1;
   RESULT = 2;
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
    QUOTIENT = 2;
    SUSPEND;
    I = 2;
    WHILE (I <= LAST_NUM) DO BEGIN
       INTERMEDIATE = RESULT;
       RESULT = PREV_ITEM + RESULT;
       PREV_ITEM = INTERMEDIATE;
       QUOTIENT = RESULT / CAST(PREV_ITEM AS DOUBLE PRECISION);
       I = I + 1;
       SUSPEND;
    END
END ^
```

Обратите внимание, что при выполнении деления для получения частного (выходной параметр QUOTIENT) один из целочисленных операндов явно при помощи функции CAST преобразуется к типу данных DOUBLE PRECISION. Это делается для того, чтобы в результате деления не были потеряны дробные знаки. Подробнее об арифметических операциях и о преобразовании данных см. в главе 3.

Каждый раз, когда встречается оператор SUSPEND, вызвавшей программе передаются очередные значения выходных параметров.

Обращение к этой процедуре можно выполнить, например, при помощи следующего оператора SELECT:

```
SELECT RESULT, CAST (QUOTIENT AS DECIMAL(18, 16))
FROM FIBONACCI_SEQ(10);
```

Здесь для дробного числа также выполняется преобразование CAST для того, чтобы получить максимальное количество дробных знаков.

Далее приведен пример простой селективной процедуры, которая выбирает из таблицы REGION строки, принадлежащие одной стране. Код страны передается процедуре в качестве входного параметра.

```
CREATE PROCEDURE PROC_SELECT_REGION2 (CODCOUNTRY CHAR(3))
RETURNS (CODREGION CHAR(4), NAMEREG CHAR(30), CENTER CHAR(15))
AS
BEGIN
   FOR SELECT CODREGION,
              NAMEREG,
              CENTER
   FROM REGION
   WHERE CODCOUNTRY = :CODCOUNTRY
   INTO :CODREGION, :NAMEREG, :CENTER
   DO SUSPEND;
END ^
```

Выборка данных осуществляется в операторе FOR SELECT-DO. Условие выборки задается в предложении WHERE, где требуется равенство кода страны значению, полученному из входного параметра процедуры. Значения столбцов очередной записи помещаются в выходные параметры. Оператор SUSPEND временно приостанавливает выполнение процедуры и передает значения выходных параметров вызвавшей программе.

Для обращения к такой процедуре можно использовать следующий оператор SELECT:

```
SELECT
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
   CODREGION AS "Код региона",
   NAMEREG AS "Название региона",
   CENTER AS "Центр региона"
FROM PROC_SELECT_REGION2 ('USA');
```

Здесь будут выбраны все штаты США.

В операторе SELECT, вызывающем хранимую процедуру, можно задавать варианты упорядоченности результатов запроса, дополнительные условия выборки. Чтобы упорядочить результат по кодам региона, можно выполнить следующий оператор:

```
SELECT
   CODREGION AS "Код региона",
   NAMEREG AS "Название региона",
   CENTER AS "Центр региона"
FROM PROC_SELECT_REGION2 ('USA')
ORDER BY CODREGION;
```

В предложении ORDER BY можно задать не только имя столбца, но и его псевдоним. Предыдущий запрос можно записать в следующем виде:

```
SELECT
   CODREGION AS "Код региона",
   NAMEREG AS "Название региона",
   CENTER AS "Центр региона"
FROM PROC_SELECT_REGION2 ('USA')
ORDER BY "Код региона";
```

Значение входного параметра (код страны) можно получить и при помощи более сложных конструкций, а не только указав строковый литерал. Следующий оператор позволяет выбрать регионы России:

```
SELECT
   CODREGION AS "Код региона",
   NAMEREG AS "Название региона",
   CENTER AS "Центр региона"
FROM PROC_SELECT_REGION2 ((SELECT CODCOUNTRY
                           FROM COUNTRY
                           WHERE NAME = 'Россия'))
ORDER BY "Код региона";
```

Здесь код страны получается при помощи оператора SELECT, обращающегося к таблице стран.

Следующий оператор SELECT позволяет при обращении к хранимой процедуре задать дополнительные условия выборки. Окончательное условие для выборки строк таблицы будет сформировано конъюнкцией (логической операцией И) условия в хранимой процедуре и условия в операторе SELECT.

```
SELECT
   CODREGION AS "Код региона",
   NAMEREG AS "Название региона",
   CENTER AS "Центр региона"
FROM PROC_SELECT_REGION2 ((SELECT CODCOUNTRY
                           FROM COUNTRY
                           WHERE NAME = 'Россия'))
WHERE NAMEREG CONTAINING 'a'
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
ORDER BY "Название региона";
```

В этом операторе выбираются только те регионы России, в названии которых присутствует буква «а». Результирующие строки упорядочиваются по названию региона.

18.7 Объявление подпроцедуры

В хранимых функциях и хранимых процедурах можно объявлять подпроцедуры. Синтаксис оператора представлен в листинге 18.6.

Листинг 18.6. Синтаксис оператора объявления подпроцедуры DECLARE PROCEDURE

```
DECLARE PROCEDURE <имя подпроцедуры>
   [(<входной параметр> [, <входной параметр> ...])]
[RETURNS (<выходной параметр> [, <выходной параметр> ...])]
AS
   [<объявление лок.переменных/курсоров>[<объявление лок.переменных/курсоров>...] ]
BEGIN
   <блок операторов>
END
```

Подпроцедура не может быть вложена в другую подпрограмму. Они поддерживаются только в основном модуле (хранимой процедуре, хранимой функции и анонимном PSQL блоке).

В настоящее время подпроцедура не имеет прямого доступа для использования переменных, курсоров и других подпрограмм из основного модуля. Это может быть разрешено в будущем.

Одна подпрограмма может вызывать другую подпрограмму, в том числе рекурсивно. В ряде случаев может потребоваться предварительное объявление подпрограммы. Общее правило: одна подпрограмма может вызвать другую подпрограмму, если последняя объявлена выше точки вызова. Все объявленные подпрограммы должны быть реализованы с той же сигнатурой. Значения по умолчанию для параметров подпрограмм не могут быть переопределены. Это означает, что они могут быть определены в реализации только тех подпрограмм, которые не были объявлены ранее.

Глава 19

Хранимые функции (FUNCTION)

Хранимая функция является программой, хранящейся в области метаданных базы данных и выполняющейся на стороне сервера. К хранимой функции могут обращаться хранимые процедуры, хранимые функции (в том числе и сама к себе), триггеры и клиентские программы. При обращении хранимой функции самой к себе такая хранимая функция называется рекурсивной.

В отличие от хранимых процедур хранимые функции всегда возвращают одно скалярное значение. Для возврата значения из хранимой функции используется оператор RETURN, который немедленно прекращает выполнение функции.

19.1 Создание хранимой функции

Для создания хранимой функции используется оператор CREATE FUNCTION, синтаксис которого представлен в листинге 19.1.

Листинг 19.1. Синтаксис оператора создания хранимой функции CREATE FUNCTION

```
CREATE FUNCTION <имя хранимой функции>
   [(<входной параметр> [, <входной параметр> ...])]
RETURNS <тип> [COLLATE <сортировка>] [DETERMINISTIC]
[SQL SECURITY {DEFINER | INVOKER}]
{EXTERNAL NAME '<внешний модуль>' ENGINE <имя движка> [AS <тело внешней функции>]}|
{
   {AS '<BLR код>'}
   |
   {AS
      [<объявление> [<объявление> ...] ]
   BEGIN
      <блок операторов>
   END }
}

<входной параметр> ::= <описание параметра> [{=|DEFAULT} <значение по умолчанию>]

<описание параметра> ::= <имя параметра> <тип> [NOT NULL]
                         [COLLATE <порядок сортировки>]

<тип> ::= <тип данных SQL>
        | [TYPE OF] <имя домена>
        | TYPE OF COLUMN <имя таблицы/представления>.<имя столбца>

<значение по умолчанию> ::= {<литерал> | NULL | <контекстная переменная>}

<внешний модуль> ::= '<имя внешнего модуля>!<имя функции в модуле>[! <информация>]'

<объявление> ::= <объявление локальной переменной>;
               | <объявление курсора>;
               | <объявление процедуры/функции>
               | <реализация процедуры/функции>
```

Таблица 19.1 — Параметры оператора CREATE FUNCTION

Параметр	Описание
Имя хранимой функции	Имя хранимой функции. Может содержать до 63 символов.
Входной параметр	Описание входного параметра.
Литерал	Литерал, совместимый по типу с параметром.
Контекстная переменная	Любая контекстная переменная, тип которой совместим с типом параметра.
Имя параметра	Имя входного параметра функции. Может содержать до 63 символов. Имя параметра должно быть уникальным среди входных параметров функции, а также её локальных переменных.
Тело внешней функции	Тело внешней функции. Строковый литерал который может использоваться UDR для различных целей.
Имя внешнего модуля	Имя внешнего модуля, в котором расположена функция.
Имя функции в модуле	Внутреннее имя функции внутри внешнего модуля.
Информация	Определяемая пользователем информация для передачи в функцию внешнего модуля.
Имя движка	Имя движка для использования внешних функций. Обычно указывается имя UDR.

Оператор CREATE FUNCTION создаёт новую хранимую функцию. Имя хранимой функции должно быть уникальным среди имён всех хранимых функций и внешних (UDF) функций. Если только это не внутренняя функция ("подпрограмма"). Для внутренних функций достаточно уникальности только в рамках модулей, которые их "охватывают".

Желательно также, чтобы имя хранимой функции было уникальным и среди имён функций расположенных в PSQL пакетах, хотя это и допустимо. Дело в том, что в настоящее время вы не сможете вызвать функцию/процедуру из глобального пространства имён внутри пакета, если в пакете объявлена одноименная функция/процедура. В этом случае всегда будет вызвана процедура/функция пакета.

CREATE FUNCTION является составным оператором, состоящий из заголовка и тела. Заголовок определяет имя хранимой функции, объявляет входные параметры и тип возвращаемого значения.

Тело функции состоит из необязательных объявлений локальных переменных, подпрограмм и именованных курсоров, и одного или нескольких операторов, или блоков операторов, заключённых во внешнем блоке, который начинается с ключевого слова BEGIN, и завершается ключевым словом END. Объявления локальных переменных и именованных курсоров, а также внутренние операторы должны завершаться точкой с запятой (;).

19.1.1 Входные параметры

Входные параметры заключаются в скобки после имени хранимой функции. Они передаются в функцию по значению, то есть любые изменения входных параметров внутри функции никак не повлияет на значения этих параметров в вызывающей программе.

У каждого параметра указывается тип данных. Кроме того, для параметра можно указать ограничение NOT NULL, тем самым запретив передавать в него значение NULL.

Для параметра строкового типа существует возможность задать порядок сортировки с помощью предложения COLLATE.

Входные параметры могут иметь значение по умолчанию. Параметры, для которых заданы зна-

чения, должны располагаться в конце списка параметров.

Использование доменов при объявлении параметров

В качестве типа параметра можно указать имя домена. В этом случае параметр будет наследовать все характеристики домена.

Если перед названием домена дополнительно используется предложение `TYPE OF`, то используется только тип данных домена - не проверяется (не используется) его ограничение (если оно есть в домене) на `NOT NULL`, `CHECK` ограничения и/или значения по умолчанию. Если домен текстового типа, то всегда используется его набор символов и порядок сортировки.

19.1.2 Использование типа столбца при объявлении параметров

Входные и выходные параметры можно объявлять, используя тип данных столбцов существующих таблиц и представлений. Для этого используется предложение `TYPE OF COLUMN`, после которого указывается имя таблицы или представления и через точку имя столбца.

При использовании `TYPE OF COLUMN` наследуется только тип данных, а в случае строковых типов ещё и набор символов, и порядок сортировки. Ограничения и значения по умолчанию столбца никогда не используются.

19.1.3 Возвращаемое значение

Предложение `RETURNS` задаёт тип возвращаемого значения хранимой функции. Если функция возвращает значение строкового типа, то существует возможность задать порядок сортировки с помощью предложения `COLLATE`. В качестве типа выходного значения можно указать имя домена, ссылку на его тип (с помощью предложения `TYPE OF`) или ссылку на тип столбца таблицы (с помощью предложения `TYPE OF COLUMN`).

19.1.4 Детерминированные функции

Необязательное предложение `DETERMINISTIC` указывает, что функция детерминированная. Детерминированные функции каждый раз возвращают один и тот же результат, если предоставлять им один и тот же набор входных значений. Недетерминированные функции могут возвращать каждый раз разные результаты, даже если предоставлять им один и тот же набор входных значений. Если для функции указано, что она является детерминированной, то такая функция не вычисляется заново, если она уже была вычислена однажды с данным набором входных аргументов, а берет свои значения из кэша метаданных (если они там есть).

19.1.5 Тело хранимой функции

После ключевого слова `AS` следует тело хранимой функции.

Объявление локальных переменных, курсоров и подпрограмм

В необязательной секции `<объявление>` описаны локальные переменные функции, именованные курсоры и подпрограммы (подпроцедуры и подфункции). Локальные переменные подчиняются тем же правилам, что и входные параметры функции в отношении спецификации типа данных.

После необязательной секции деклараций обязательно следует составной оператор. Составной оператор состоит из одного или нескольких `PSQL` операторов, заключенных между ключевыми словами `BEGIN` и `END`. Составной оператор может содержать один или несколько других составных операторов. Вложенность ограничена 512 уровнями. Любой из `BEGIN .. END` блоков может быть пустым, в том числе и главный блок.

19.1.6 Внешние функции

Хранимая функция может быть расположена во внешнем модуле. В этом случае вместо тела функции указывается место расположения функции во внешнем модуле с помощью предложения `EXTERNAL NAME`. Аргументом этого предложения является строка, в которой через разделитель указано имя внешнего модуля, имя функции внутри модуля и определённая пользователем информация. В предложении `ENGINE` указывается имя движка для обработки подключения внешних модулей. В РЕД Базе Данных для работы с внешними модулями используется движок `UDR`. После ключевого слова `AS` может быть указан строковый литерал — "тело" внешней функции, оно может быть использовано внешнем модулем для различных целей.

Не следует путать внешние функции, объявленные как `DECLARE EXTERNAL FUNCTION`, так же известные как `UDF`, с функциями расположенными во внешних модулях объявленных как `CREATE FUNCTION ... EXTERNAL NAME`, называемых `UDR` (`User Defined Routine`). Первые являются унаследованными (`Legacy`) из предыдущих версий РЕД Базы Данных. Их возможности существенно уступают возможностям нового типа внешних функций. В РЕД Базе Данных 5.0 `UDF` объявлены устаревшими.

19.1.7 Привилегии выполнения

Необязательное предложение `SQL SECURITY {DEFINER | INVOKER}` определяет, в контексте какого пользователя будет выполняться функция. Ключевое слово `INVOKER` (значение по умолчанию) указывает, что функция выполняется с правами вызвавшего ее пользователя. Задание ключевого слова `DEFINER` означает, что функция выполняется с правами к объектам базы данных ее владельца (создателя). Значение по умолчанию на уровне всей базы данных можно изменить оператором `ALTER DATABASE SET DEFAULT SQL SECURITY`.

19.1.8 Использование BLR вместо SQL кода

В РЕД Базе Данных 5.0 появилась возможность создать функцию с телом в виде двоичного `BLR` представления, закодированного в `Base64`, вместо обычного исходного кода на языке `SQL`. В этом случае исходные `SQL` коды функции будут недоступны. Оператор создания такой функции будет выглядеть так:

```
CREATE FUNCTION <имя> (<входные параметры>)
RETURNS <тип>
[SQL SECURITY {DEFINER | INVOKER}]
AS '<BLR код>';
```

Здесь `<BLR код>` - это BLR в его исходном двоичном виде, закодированный в `Base64`. Его можно получить, например, из запроса:

```
select BASE64_ENCODE(RDB$FUNCTION_BLR)
from RDB$FUNCTIONS
where RDB$FUNCTION_NAME = '<имя>';
```

19.2 Изменение хранимой функции

Для изменения существующей хранимой функции используется оператор `ALTER FUNCTION`. Синтаксис оператора представлен в листинге 19.2.

Листинг 19.2. Синтаксис оператора изменения хранимой функции ALTER FUNCTION

```
ALTER FUNCTION <имя хранимой функции>
   [(<входной параметр> [, <входной параметр> ...])]
RETURNS <тип> [COLLATE <сортировка>] [DETERMINISTIC]
[SQL SECURITY {DEFINER | INVOKER}]
{ EXTERNAL NAME '<внешний модуль>' ENGINE <имя движка> [AS <тело внешней функции>]}
|
{
   {AS '<BLR код>'}
   |
   {AS
      [<объявление> [<объявление> ...] ]
   BEGIN
      <блок операторов>
   END }
}
```

Изменять хранимую функцию может администратор, владелец хранимой функции, пользователь с привилегией ALTER ANY FUNCTION.

Оператор ALTER FUNCTION позволяет изменять:

- состав и характеристики входных параметров;
- тип выходного значения;
- локальные переменные, курсоры, подпрограммы;
- в контексте какого пользователя будет выполняться функция;
- тело хранимой функции на языке SQL;
- исходный SQL код функции на двоичный BLR код;
- точку входа и имя движка (для внешних функций)

После выполнения существующие привилегии и зависимости сохраняются.

Внешние функции, объявленные как DECLARE EXTERNAL FUNCTION, так же известные как UDF, невозможно преобразовать в PSQL функции и наоборот.

Будьте осторожны при изменении количества и типов входных параметров хранимых функций. Существующий код приложения может стать неработоспособным из-за того, что формат вызова функции несовместим с новым описанием параметров. Кроме того, PSQL модули, использующие изменённую хранимую функцию, могут стать некорректными. Информация о том, как это обнаружить, находится в разделе Приложение B.

Если у вас уже есть внешняя функция в Legacy стиле (DECLARE EXTERNAL FUNCTION), то оператор ALTER FUNCTION изменит её на обычную функцию без всяких предупреждений. Это было сделано умышлено для облегчения миграции на новый стиль написания внешних функций известных как UDR.

Параметр DETERMINISTIC можно изменять без указания тела функции:

```
ALTER FUNCTION <имя хранимой функции> {DETERMINISTIC | NOT DETERMINISTIC}
```

Изменить значение SQL SECURITY можно без указания тела функции:

```
ALTER FUNCTION <имя хранимой функции>
  SQL SECURITY {DEFINER | INVOKER}
| DROP SQL SECURITY
```

19.3 Создание новой или изменение существующей хранимой функции

Оператор `CREATE OR ALTER FUNCTION` позволяет создать новую хранимую функцию, если функция с тем же именем отсутствует в базе данных, или изменить описание существующей в базе данных функции. Если функция с этим именем уже существует, то происходит ее замена на новую хранимую функцию, при этом существующие привилегии и зависимости сохраняются. Синтаксис оператора представлен в листинге 19.3.

Листинг 19.3. Синтаксис оператора создания новой или изменения существующей хранимой функции `CREATE OR ALTER FUNCTION`

```
CREATE OR ALTER FUNCTION <имя хранимой функции>
   [(<входной параметр> [, <входной параметр> ...])]
RETURNS <тип> [COLLATE <сортировка>] [DETERMINISTIC]
[SQL SECURITY {DEFINER | INVOKER}]
{ EXTERNAL NAME '<внешний модуль>' ENGINE <имя движка> [AS <тело внешней функции>]}
|
{
   {AS '<BLR код>'}
   |{AS
      [<объявление> [<объявление> ...] ]
   BEGIN
      <блок операторов>
   END }
}
```

19.4 Удаление хранимой функции

Для удаления существующей хранимой функции используется оператор `DROP FUNCTION`. Синтаксис оператора представлен в листинге 19.4.

Листинг 19.4. Синтаксис оператора удаления хранимой функции `DROP FUNCTION`

```
DROP FUNCTION <имя хранимой процедуры>
```

Если от хранимой функции существуют зависимости, то при попытке удаления такой функции будет выдана соответствующая ошибка.

Удалить хранимую функцию может администратор, владелец хранимой функции и пользователь с привилегией `DROP ANY FUNCTION`.

19.5 Создание новой или пересоздание существующей хранимой функции

Оператор RECREATE FUNCTION позволяет создать новую хранимую функцию, если функция с тем же именем отсутствует в базе данных, или пересоздать существующую в базе данных функцию. Если функция с этим именем уже существует, то оператор попытается удалить её и создать новую функцию, при этом существующие привилегии и зависимости не сохраняются. Синтаксис оператора представлен в листинге 19.5.

Листинг 19.5. Синтаксис оператора создания новой или изменения существующей хранимой функции RECREATE FUNCTION

```
RECREATE FUNCTION <имя хранимой функции>
   [(<входной параметр> [, <входной параметр> ...])]
RETURNS <тип> [COLLATE <сортировка>] [DETERMINISTIC]
[SQL SECURITY {DEFINER | INVOKER}]
{ EXTERNAL NAME '<внешний модуль>' ENGINE <имя движка> [AS <тело внешней функции>]}
|
{
   {AS '<BLR код>'}
   |
   {AS
      [<объявление> [<объявление> ...] ]
   BEGIN
      <блок операторов>
   END }
}
```

Операция закончится неудачей при подтверждении транзакции, если функция имеет зависимости.

19.6 Объявление подфункции

В хранимых функциях и хранимых процедурах можно объявлять подфункции. Синтаксис оператора представлен в листинге 19.6.

Листинг 19.6. Синтаксис оператора объявления подфункции DECLARE FUNCTION

```
DECLARE FUNCTION <имя подфункции> [(<входной> [, <входной> ...])]
      RETURNS <тип> [COLLATE <сортировка>] [DETERMINISTIC];
```

Подфункция не может быть вложена в другую подпрограмму. Они поддерживаются только в основном модуле (хранимой процедуре, хранимой функции и анонимном PSQL блоке).

Подпроцедура может использовать переменные, но не курсоры, из своего родительского модуля. Она может обращаться к другим процедурам из своих родительских модулей.

Одна подпрограмма может вызывать и другую подпрограмму, в том числе рекурсивно. В ряде случаев может потребоваться предварительное объявление подпрограммы. Общее правило: одна подпрограмма может вызвать другую подпрограмму, если последняя объявлена выше точки вызова. Все объявленные подпрограммы должны быть реализованы с той же сигнатурой. Значения по умолчанию для параметров подпрограмм не могут быть переопределены. Это означает, что они могут быть определены в реализации только тех подпрограмм, которые не были объявлены ранее.

Глава 20

Триггеры (TRIGGER)

Триггер является программой, которая хранится в области метаданных базы данных и выполняется на стороне сервера. Напрямую обращение к триггеру невозможно. Он вызывается автоматически при наступлении одного или нескольких событий, относящихся к одной конкретной таблице (к представлению), или при наступлении одного из событий базы данных. Триггер, вызываемый при наступлении события таблицы, связан с одной таблицей или представлением, с одним или более событиями для этой таблицы или представления (добавление, изменение или удаление данных) и ровно с одной фазой такого события (до наступления события или после этого). Триггер выполняется в контексте той транзакции, в контексте которой выполнялась программа, вызвавшая соответствующее событие. Исключением являются триггеры, реагирующие на события базы данных. Для некоторых из них запускается транзакция по умолчанию.

В СУБД «РЕД База Данных» различают три вида триггеров в зависимости от событий, на которые они реагируют:

- Табличные или DML триггеры;
- Триггеры на события базы данных;
- Триггеры на события изменения метаданных или DDL триггеры.

Порядок выполнения

Для каждой комбинации фазы и события может быть определено несколько триггера. Порядок их выполнения может быть указан в явном виде с помощью необязательного предложения POSITION в определении триггера. Всего доступно 32 767 позиций. Триггеры с наименьшими значениями позиций запускаются первыми. Если аргумент POSITION не указан, то по умолчанию позиция равна 0. Если несколько триггеров имеют одинаковую позицию и фазу, то они будут выполняться в неопределенном порядке, при этом соблюдается общий порядок по позиции и фазе.

DML триггеры

DML триггеры вызываются при изменении состояния данных DML операциями: редактирование, добавление или удаление строк. Они могут быть определены и для таблиц и для представлений.

Существует шесть вариантов соотношения событие-фаза для таблицы (представления):

- до добавления новой строки (BEFORE INSERT);
- после добавления новой строки (AFTER INSERT);
- до изменения строки (BEFORE UPDATE);
- после изменения строки (AFTER UPDATE);
- до удаления строки (BEFORE DELETE);
- после удаления строки (AFTER DELETE).

Эти базовые формы предназначены для создания триггеров с одной фазой и одним событием. Также поддерживаются формы для создания триггеров для одной фазы и нескольких событий, например, BEFORE INSERT OR UPDATE OR DELETE или AFTER UPDATE OR DELETE. Многофазные триггеры, такие как BEFORE OR AFTER, невозможны. Логические контекстные переменные INSERTING, UPDATING и DELETING можно использовать в теле триггера для определения типа события, вызвавшего срабатывание триггера.

Контекстные переменные OLD и NEW

Для триггеров DML существуют контекстные переменные OLD и NEW. Каждая из них содержит запись значений всей строки: одна для значений до события изменения данных (фаза BEFORE), другая - для значений, которые будут после события (фаза AFTER). В операторах на эти контекстные переменные ссылаются, используя форму NEW.<имя столбца> и OLD.<имя столбца> соответственно. В триггерах можно обращаться к значению любого столбца таблицы (представления) до его изменения в клиентской программе (для этого перед именем столбца помещается ключевое слово OLD и точка) и после изменения (перед именем столбца помещается NEW и точка).

Контекстная переменная OLD является переменной только для чтения для всех видов триггеров. Она недоступна в триггерах, вызываемых при добавлении данных (INSERT), независимо от фазы события.

Контекстная переменная NEW в триггерах для фазы события после (AFTER) является переменной только для чтения. Она недоступна в триггерах для события удаления данных.

Для BEFORE UPDATE и BEFORE INSERT переменная NEW доступна для чтения и записи, только если не затрагивает вычисляемый столбец COMPUTED BY.

Триггеры на события базы данных

События базы данных, для которых может быть определён триггер:

Таблица 20.1 — Триггеры на события базы данных

Событие описание	Описание
CONNECT	Перед выполнением триггера автоматически запускается транзакция с уровнем изоляции по умолчанию.
DISCONNECT	Перед выполнением триггера автоматически запускается транзакция с уровнем изоляции по умолчанию.
TRANSACTION START	Триггер выполняется в контексте текущей транзакции.
TRANSACTION COMMIT	Триггер выполняется в контексте транзакции, выполняющей фиксацию.
TRANSACTION ROLLBACK	Триггер выполняется в контексте транзакции, выполняющей откат.

Триггеры на события CONNECT и DISCONNECT выполняются в специально созданной для этого транзакции. Если при обработке триггера не было вызвано исключение, то транзакция подтверждается. Необработанные исключения откатят транзакцию. В этом случае у триггера на событие CONNECT соединение разрывается, а исключения возвращается клиенту. Для триггера на событие DISCONNECT соединение разрывается, но исключения не возвращается клиенту.

Триггеры на событие TRANSACTION срабатывают при старте транзакции, её фиксации и откате. Необработанные исключения обрабатываются в зависимости от типа события TRANSACTION:

- для события TRANSACTION START исключение возвращается клиенту, а транзакция отменяется;
- для события TRANSACTION COMMIT исключение возвращается клиенту, действия, выполненные триггером, и транзакция отменяются;
- для события TRANSACTION ROLLBACK исключение не возвращается клиенту, а транзакция отменяется.

Из вышеизложенного следует, что нет прямого способа узнать, какой триггер (DISCONNECT или ROLLBACK) вызвал исключение. Также ясно, что вы не сможете подключиться к базе данных в случае исключения в триггере на событие CONNECT, а также отменяется старт транзакции при исключении в триггере на событие TRANSACTION START. В обоих случаях база данных блокируется до тех пор, пока вы не отключите триггеры базы данных и не исправите ошибочный код.

В некоторые утилиты командной строки были добавлены новые ключи для отключения триггеров

на базу данных:

```
gbak -nodbtriggers
isql -nodbtriggers
nbackup -T
```

В случае двухфазных транзакций триггеры на событие TRANSACTION START срабатывают в фазе подготовки (prepare), а не в фазе commit.

Использование оператора IN AUTONOMOUS TRANSACTION DO в триггерах событий базы данных, связанных с транзакциями (TRANSACTION START, TRANSACTION ROLLBACK, TRANSACTION COMMIT), может привести к тому, что автономная транзакция попадет в бесконечный цикл.

Триггеры событий DISCONNECT и TRANSACTION ROLLBACK не будут выполняться при отключении клиентов через таблицы мониторинга (DELETE FROM MON$ATTACHMENTS).

DDL триггеры

Триггеры на события изменения метаданных (DDL триггеры) предназначены для обеспечения ограничений, которые будут распространены на пользователей, которые пытаются создать, изменить или удалить DDL объект. Другое их назначение — ведение журнала изменений метаданных.

DDL триггеры срабатывают на указанные события изменения метаданных в одной из фаз события. BEFORE триггеры запускаются до изменений в системных таблицах, AFTER триггеры запускаются после изменений в системных таблицах.

Тип события [BEFORE | AFTER] для триггера DDL изменить нельзя.

Когда оператор DDL запускает триггер, в котором возбуждается исключение, оператор не будет фиксирован. Т.е. исключения могут использоваться, чтобы гарантировать, что оператор DDL будет отменен, если некоторые условия не будут соблюдены.

Действия DDL триггеров выполняются только при фиксации транзакции, в которой работает затронутая DDL команда. Никогда не забывайте о том, что в AFTER триггере, возможно сделать только то, что возможно сделать после DDL команды без автоматической фиксации транзакций.

Для операторов CREATE OR ALTER... триггер срабатывает один раз для события CREATE или события ALTER, в зависимости от того существовал ли ранее объект. Для операторов RECREATE триггер вызывается для события DROP, если объект существовал, и после этого для события CREATE.

Если объект метаданных не существует, то обычно триггеры на события ALTER и DROP не запускаются. Исключением из правила являются BEFORE ALTER/DROP USER триггеры, которые будут вызваны, даже если имя пользователя не существует. Это вызвано тем, что эти команды выполняются для базы данных безопасности, для которой не делается проверка существования пользователей перед их выполнением. Данное поведение, вероятно, будет отличаться для встроенных пользователей.

Если некоторое исключение возбуждено после того как начала выполняться DDL команда и до того как запущен AFTER триггер, то AFTER триггер не запускается.

Для процедур и функций в составе пакетов не запускаются индивидуальные триггеры {CREATE | ALTER | DROP} {PROCEDURE | FUNCTION}.

Оператор ALTER DOMAIN <старое имя> TO <новое имя> устанавливает контекстные переменные OLD_OBJECT_NAME и NEW_OBJECT_NAME в обоих триггерах BEFORE и AFTER. Контекстная переменная OBJECT_NAME будет содержать старое имя объекта метаданных в триггере BEFORE, и новое — в триггере AFTER.

Если в качестве события указано предложение ANY DDL STATEMENT, то триггер будет вызван при наступлении любого из DDL событий.

Пространство имён DDL_TRIGGER

Во время работы DDL триггера доступно пространство имён DDL_TRIGGER для использования в функции RDB$GET_CONTEXT. Его использование также допустимо в хранимых процедурах и функциях, вызванных DDL триггерами.

Контекст DDL_TRIGGER работает как стек. Перед возбуждением DDL триггера, значения, относящиеся к выполняемой команде, помещаются в этот стек. После завершения работы триггера значения выталкиваются. Таким образом, в случае каскадных DDL операторов, когда каждая пользовательская DDL команда возбуждает DDL триггер, и этот триггер запускает другие DDL команды, с помощью EXECUTE STATEMENT, значения переменных в пространстве имен DDL_TRIGGER будут соответствовать команде, которая вызвала последний DDL триггер в стеке вызовов.

Переменные доступные в пространстве имён DDL_TRIGGER:

- EVENT_TYPE — тип события (CREATE, ALTER, DROP);
- OBJECT_TYPE — тип объекта (TABLE, VIEW и д.р.);
- DDL_EVENT — имя события (EVENT_TYPE || ' ' || OBJECT_TYPE);
- OBJECT_NAME — имя объекта метаданных;
- SQL_TEXT — текст SQL-запроса.

20.1 Создание триггера

Для создания триггера используется оператор CREATE TRIGGER, синтаксис которого представлен в листинге 20.1.

Листинг 20.1. Синтаксис оператора создания триггера CREATE TRIGGER

```
CREATE TRIGGER <имя триггера> {
      <объявление табличного триггера>
    | <объявление табличного триггера в стандарте SQL-2003>
    | <объявление триггера базы данных>
    | <объявление DDL триггера> }
[SQL SECURITY {DEFINER | INVOKER}]
{ EXTERNAL NAME '<внешний модуль>' ENGINE <имя движка> [AS <тело внеш. триг.>]} |
{
   AS
      [<объявление> [<объявление> ...] ]
   BEGIN
      <блок операторов>
   END }

<объявление табличного триггера> ::=
         FOR {<имя таблицы> | <имя представления>}
         [ACTIVE | INACTIVE]
         {BEFORE | AFTER} <список событий таблицы (представления)>
         [POSITION <порядок срабатывания триггера>]

<объявление табличного триггера в стандарте SQL-2003> ::=
         [ACTIVE | INACTIVE]
         {BEFORE | AFTER} <список событий таблицы (представления)>
         ON {<имя таблицы> | <имя представления>}
         [POSITION <порядок срабатывания триггера>]

<объявление триггера базы данных> ::=
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
          [ACTIVE | INACTIVE]
          ON <событие соединения или транзакции>
          [POSITION <порядок срабатывания триггера>]

<объявление DDL триггера> ::=
          [ACTIVE | INACTIVE]
          {BEFORE | AFTER} <список DDL событий>
          [POSITION <порядок срабатывания триггера>]

<список событий таблицы (представления)> ::= <событие DML> [OR <событие DML>...]

<событие DML> ::= { INSERT | UPDATE | DELETE }

<событие соединения или транзакции> ::= {
            CONNECT
          | DISCONNECT
          | TRANSACTION START
          | TRANSACTION COMMIT
          | TRANSACTION ROLLBACK }

<список DDL событий> ::= {
            <любой оператор DDl>
          | <DDL событие> [OR <DDL событие> ...] }

<DDL событие> ::=
            CREATE|ALTER|DROP TABLE
          | CREATE|ALTER|DROP PROCEDURE
          | CREATE|ALTER|DROP FUNCTION
          | CREATE|ALTER|DROP TRIGGER
          | CREATE|ALTER|DROP EXCEPTION
          | CREATE|ALTER|DROP VIEW
          | CREATE|ALTER|DROP DOMAIN
          | CREATE|ALTER|DROP ROLE
          | CREATE|ALTER|DROP SEQUENCE
          | CREATE|ALTER|DROP USER
          | CREATE|ALTER|DROP INDEX
          | CREATE|DROP COLLATION
          | ALTER CHARACTER SET
          | CREATE|ALTER|DROP PACKAGE
          | CREATE|DROP PACKAGE BODY
          | CREATE|ALTER|DROP MAPPING

<объявление> ::= <объявление локальной переменной>;
                 | <объявление курсора>;
                 | <объявление процедуры/функции>
                 | <реализация процедуры/функции>
```

Оператор `CREATE TRIGGER` используется для создания нового триггера. Триггер может быть создан либо для события таблицы/представления (или комбинации событий), для события базы данных и для события `DDL`.

Оператор `CREATE TRIGGER` является составным оператором, содержащими заголовок и тело. Заголовок определяет имя триггера, а также содержит имя таблицы/представления, фазу триггера, событие (или события) на которые срабатывает триггер и позицию. Тело триггера состоит из необя-

зательных объявлений локальных переменных, подпрограмм и именованных курсоров, и одного или нескольких операторов или блоков операторов, заключённых во внешнем блоке, который начинается с ключевого слова BEGIN и заканчивается ключевым словом END. Объявления и внутренние операторы завершаются точкой с запятой (;).

Имя триггера может содержать до 63 символов и должно быть уникальным среди имен всех триггеров базы данных.

20.1.1 Состояние триггера

Триггер может быть активным (ACTIVE) или неактивным (INACTIVE). Если триггер активен (значение по умолчанию), то он автоматически вызывается при наступлении соответствующего события (событий) таблицы или базы данных. Если триггер неактивен, то вызов триггера не происходит.

20.1.2 Внешние триггеры

Триггер может быть расположена во внешнем модуле. В этом случае вместо тела триггера указывается место его расположения во внешнем модуле с помощью предложения EXTERNAL NAME. Аргументом этого предложения является строка, в которой через разделитель указано имя внешнего модуля, имя процедуры внутри модуля и определённая пользователем информация. В предложении ENGINE указывается имя движка для обработки подключения внешних модулей. В РЕД Базе Данных для работы с внешними модулями используется движок UDR. После ключевого слова AS может быть указан строковый литерал — "тело" внешнего триггера, оно может быть использовано внешнем модулем для различных целей.

20.1.3 Привилегии выполнения

Необязательное предложение SQL SECURITY {DEFINER | INVOKER} определяет, в контексте какого пользователя будет выполняться триггер. Ключевое слово INVOKER (значение по умолчанию) указывает, что триггер выполняется с правами вызвавшего его пользователя. Задание ключевого слова DEFINER означает, что триггер выполняется с правами к объектам базы данных его владельца (создателя). Значение по умолчанию на уровне всей базы данных можно изменить оператором ALTER DATABASE SET DEFAULT SQL SECURITY.

Если для таблицы будет изменена опция SQL SECURITY, имеющиеся триггеры без явно указанной опции не будут сразу использовать новое значение, оно вступит в силу в следующий раз, когда триггер будет загружен в кэш метаданных.

20.1.4 DML триггеры

Табличный триггер может быть создан владельцем таблицы (представления), для которой создается DML триггер, администратором и пользователем с привилегией ALTER ANY {TABLE|VIEW}.

Фаза

Фаза определяет время срабатывания триггера по отношению к операции изменения данных:

- Триггер BEFORE срабатывает перед операцией изменения данных (вставка, обновление или удаление).
- Триггер AFTER запускается после завершения операции изменения данных.

Порядок срабатывания

Для одного и того же события или группы событий может быть создано несколько триггеров. Ключевое слово POSITION позволяет задать порядок, в котором будут выполняться такие триггеры (по умолчанию значение 0). Если несколько триггеров имеют одинаковые позицию и фазу, эти триггеры будут выполняться в неопределенном порядке, при этом соблюдая общий порядок по позиции и фазе.

События строк

В определении триггера таблицы/представления указывается хотя бы одна из DML-операций (INSERT, UPDATE и DELETE), чтобы указать одно или несколько событий, на которые должен сработать триггер. Если указано несколько операций, они должны быть разделены ключевым словом OR. Ни одна операция не может выполняться более одного раза. Внутри блока операторов можно использовать логические контекстные переменные INSERTING, UPDATING и DELETING для проверки того, какая операция выполняется в данный момент.

20.1.5 Триггеры базы данных

Триггеры на события базы данных и на события изменения метаданных может создавать только владелец базы данных, администратор и пользователь с привилегией ALTER DATABASE.

20.2 Изменение триггера

Для изменения заголовка и/или тела существующего триггера используется оператор ALTER TRIGGER, синтаксис которого представлен в листинге 20.2.

Листинг 20.2. Синтаксис оператора изменения триггера ALTER TRIGGER

```
ALTER TRIGGER <имя триггера>
[ACTIVE | INACTIVE]
[{BEFORE | AFTER} <список событий таблицы (представления)>]
[POSITION <порядок срабатывания триггера>]
[SQL SECURITY {DEFINER | INVOKER} | DROP SQL SECURITY]
[{EXTERNAL NAME '<внешний модуль>' ENGINE <имя движка> [AS <тело внеш. триг.>]} |
{
   AS
      [<объявление> [<объявление> ...] ]
   BEGIN
      <блок операторов>
   END }]

<список событий таблицы (представления)> ::= <событие DML> [OR <событие DML>...]

<событие DML> ::= { INSERT | UPDATE | DELETE }
```

DML триггер может быть изменен администратором и владельцем таблицы или представления или пользователем с привилегией ALTER ANY {TABLE | VIEW}.

Триггеры для событий базы данных и триггеры событий на изменение метаданных может изменить администратор, владелец базы данных или пользователь с привилегией ALTER DATABASE.

В операторе изменения триггера можно изменить его состояние активности (ACTIVE / INACTIVE), событие (события) таблицы (представления) и фазу события, позицию триггера, выполняемые триг-

гером действия, а также в контексте какого пользователя будет выполняться триггер. Можно удалить опцию SQL SECURITY, указанную при создании.

Если триггер был создан для события базы данных или для события изменения метаданных, то можно изменить только его активность, позицию и выполняемые действия. Если какой-то элемент не указан, то его первоначальное значение не изменяется.

Триггер DML нельзя изменить на триггер базы данных или триггер DDL.

В этом операторе нельзя изменить ссылку на таблицу или представление, с которым связан существующий триггер, а также событие базы данных.

20.3 Создание нового или изменение существующего триггера

Оператор CREATE OR ALTER TRIGGER создает новый триггер для таблицы или представления, если триггера с таким именем не существует в базе данных. Иначе изменяет и перекомпилирует его, при этом существующие права и зависимости сохраняются.

Листинг 20.3. Синтаксис оператора создания нового или изменения существующего триггера CREATE OR ALTER TRIGGER

```
CREATE OR ALTER TRIGGER <имя триггера> {
   <объявление табличного триггера>
 | <объявление табличного триггера в стандарте SQL-2003>
 | <объявление триггера базы данных>
 | <объявление DDL триггера> }
[SQL SECURITY {DEFINER | INVOKER}]
{ EXTERNAL NAME '<внешний модуль>' ENGINE <имя движка> [AS <тело внеш. триг.>] } |
{
   AS
      [<объявление> [<объявление> ...] ]
   BEGIN
      <блок операторов>
   END }
```

Синтаксис этого оператора соответствует синтаксису оператора CREATE TRIGGER.

20.4 Удаление триггера

Для удаления существующего триггера используется оператор DROP TRIGGER, синтаксис которого представлен в листинге 20.4.

Листинг 20.4. Синтаксис оператора удаления триггера DROP TRIGGER

```
DROP TRIGGER <имя триггера>;
```

DML триггер может быть удален администратором и владельцем таблицы или представления или пользователем с привилегией ALTER ANY {TABLE | VIEW}.

Триггеры для событий базы данных и триггеры событий на изменение метаданных может удалить администратор, владелец базы данных или пользователь с привилегией ALTER DATABASE.

Нельзя удалить триггер, автоматически созданный системой для поддержания ограничений PRIMARY KEY, CHECK и FOREIGN KEY. Остальные триггеры не имеют никаких зависимостей, которые ограничили бы возможности удаления триггеров. Ошибки зависимостей не обнаруживаются до фиксации этой операции (COMMIT).

20.5 Создание нового или пересоздание существующего триггера

Оператор RECREATE TRIGGER создаёт новый триггер, если триггер с указанным именем не существует, в противном случае оператор RECREATE TRIGGER попытается удалить его и создать новый.

Листинг 20.5. Синтаксис оператора создания нового или пересоздания существующего триггера RECREATE TRIGGER

```
RECREATE TRIGGER <имя триггера> {
   <объявление табличного триггера>
 | <объявление табличного триггера в стандарте SQL-2003>
 | <объявление триггера базы данных>
 | <объявление DDL триггера> }
[SQL SECURITY {DEFINER | INVOKER}]
{ EXTERNAL NAME '<внешний модуль>' ENGINE <имя движка> [AS <тело внеш. триг.>] } |
{
   AS
      [<объявление> [<объявление> ...] ]
   BEGIN
      <блок операторов>
   END }
```

Если указанный триггер уже используется, то операция завершится неудачей при фиксации (COMMIT). Ошибки зависимостей не обнаруживаются до фиксации этой операции (COMMIT).

Синтаксис этого оператора соответствует синтаксису оператора CREATE TRIGGER.

20.6 Примеры триггеров

Триггеры являются полезным инструментом при выполнении различных действий в случае изменения данных базы данных. Основным назначением триггеров является автоматическое выполнение функций по формированию значений искусственного первичного ключа, выдачи информационных сообщений базы данных, связанных с изменениями данных, специфические способы поддержания декларативной целостности данных и выполнение определенных действий при добавлении (изменении) некоторых данных базы данных. Триггеры также могут быть использованы при наступлении событий, связанных с базой данных в целом — соединение с базой данных, отсоединение от базы данных, запуск, подтверждение или отмена транзакций.

20.6.1 Формирование значения искусственного первичного ключа

Одно из основных назначений триггеров — формирование значения искусственных первичных ключей в таблицах. Такие триггеры вызываются до помещения новой строки таблицы в базу данных (BEFORE INSERT).

Пусть имеется таблица PEOPLE, описывающая людей в базе данных. В этой таблице присутствует столбец COD, являющийся искусственным первичным ключом. Для получения значения этого ключа

в базе данных создан генератор с именем GEN_PEOPLE. Чтобы при операции добавления новой строки в эту таблицу получать уникальное значение искусственного первичного ключа, нужно создать следующий триггер:

```
SET TERM ^;
CREATE TRIGGER TBI_PEOPLE
   FOR PEOPLE
   ACTIVE
   BEFORE INSERT
AS BEGIN
   IF (NEW.COD IS NULL) THEN
      NEW.COD = NEXT VALUE FOR GEN_PEOPLE;
END ^
```

Триггер является активным (ACTIVE), создается для таблицы PEOPLE для фазы до (BEFORE) события добавления новой записи (INSERT).

В теле триггера проверяется, не присвоено ли уже первичному ключу какое-либо значение (стандартная проверка). Для этого используется имя столбца с префиксом NEW. После этой проверки, если первичный ключ не имеет еще никакого значения, значению первичного ключа присваивается уникальное значение, получаемое из генератора GEN_PEOPLE увеличением на единицу значения генератора.

Вместо конструкции NEXT VALUE FOR можно использовать и встроенную функцию GEN_ID:

```
NEW.COD = GEN_ID(GEN_PEOPLE, 1);
```

20.6.2 Передача сообщений клиентским процессам об изменении данных

Следующий пример триггера позволяет выдать сообщение о событии базы данных при внесении любых изменений (добавление, изменение, удаление) в таблицу стран COUNTRY. Текст триггера:

```
SET TERM ^;
CREATE TRIGGER TAC_COUNTRY
   FOR COUNTRY
   AFTER INSERT OR UPDATE OR DELETE
AS BEGIN
   POST_EVENT 'COUNTRY_CHANGED';
END ^
```

Триггер создается для таблицы COUNTRY для фазы после (AFTER) событий добавления, изменения и удаления (INSERT, UPDATE, DELETE). Если транзакция, в контексте которой выполнялись соответствующие операторы, будет подтверждена (будет выполнен оператор COMMIT или COMMIT RETAINING), то все клиентские приложения, которые прослушивают это сообщение, получат соответствующий сигнал. При отмене такой транзакции (оператор ROLLBACK) сообщение клиентам передано не будет.

Как правило, реакцией клиентов на подобное сообщение является как минимум переоткрытие соответствующего набора данных, а в некоторых случаях и перезапуск транзакции (транзакций с уровнями изоляции SNAPSHOT и SNAPSHOT TABLE STABILITY).

20.6.3 Пример триггера, обеспечивающего поддержание ссылочной целостности данных

Если при объявлении внешнего ключа в описании ограничения REFERENCES для операции UPDATE используется вариант NO ACTION, клиентская программа сама должна обеспечить соответствие внешнего ключа дочерней таблицы изменившемуся значению первичного ключа записи родительской таблицы.

Следующий триггер TAU_COUNTRY выполняет все необходимые действия по поддержанию соответствия внешних ключей подчиненной таблицы регионов (REGION) первичному ключу таблицы стран (COUNTRY) при изменении значения первичного ключа (код страны) в справочнике стран.

```
SET TERM ^;
RECREATE TRIGGER TAU_COUNTRY
   FOR COUNTRY
   AFTER UPDATE
AS BEGIN
   IF (OLD.CODCOUNTRY <> NEW.CODCOUNTRY) THEN
      UPDATE REGION
         SET REGION.CODCOUNTRY = NEW.CODCOUNTRY
         WHERE REGION.CODCOUNTRY = OLD.CODCOUNTRY;
END ^
```

Триггер вызывается после изменения строки таблицы стран. В операторе IF проверяется, изменялся ли код страны. Только в этом случае выполняются соответствующие изменения в подчиненной таблице регионов REGION — кодам страны всех подчиненных регионов присваивается измененное значение кода страны.

Похожие действия нужно выполнить для соответствующих строк подчиненной таблицы при удалении строки главной таблицы, если в описании ограничения внешнего ключа подчиненной таблицы для DELETE было задано NO ACTION. В этом случае нужно написать триггер после удаления строки главной таблицы, в котором удаляются и все зависимые строки подчиненной таблицы.

```
CREATE OR ALTER TRIGGER TAD_COUNTRY
   FOR COUNTRY
   AFTER DELETE
AS BEGIN
   DELETE FROM REGION
   WHERE REGION.CODCOUNTRY = OLD.CODCOUNTRY;
END ^
```

20.6.4 Пример триггера, создающего запись истории окладов

В демонстрационной базе данных EMPLOYEE присутствуют две таблицы: таблица EMPLOYEE, описывающая сотрудников организации, и таблица SALARY_HISTORY, хранящая историю окладов сотрудников. Операторы создания этих таблиц представлены в следующем примере:

```
CREATE TABLE EMPLOYEE (
   emp_no SMALLINT NOT NULL PRIMARY KEY,
   first_name VARCHAR(15) NOT NULL,
   last_name VARCHAR(20) NOT NULL,
   phone_ext VARCHAR(4),
   hire_date DATE DEFAULT 'NOW' NOT NULL,
   dept_no CHAR(3),
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
    job_code VARCHAR(5) NOT NULL,
    job_grade SMALLINT NOT NULL,
    job_country VARCHAR(15) NOT NULL,
    salary NUMERIC(10,2) NOT NULL,
    full_name COMPUTED BY (last_name || ', ' || first_name));

CREATE TABLE SALARY_HISTORY (
    emp_no SMALLINT NOT NULL,
    change_date DATE DEFAULT 'NOW' NOT NULL,
    updater_id VARCHAR(20) NOT NULL,
    old_salary NUMERIC(10,2) NOT NULL,
    percent_change DOUBLE PRECISION
       DEFAULT 0
       NOT NULL
       CHECK (percent_change between -50 and 50),
    new_salary COMPUTED BY (old_salary + old_salary * percent_change / 100),
    PRIMARY KEY (emp_no, change_date, updater_id),
    FOREIGN KEY (emp_no) REFERENCES employee (emp_no)  );
```

Следующий триггер вызывается после изменения (`AFTER UPDATE`) таблицы `EMPLOYEE`. В нем проверяется, не изменился ли оклад сотрудника, и если изменился, в триггере создается новая запись истории сотрудника.

```
CREATE TRIGGER SAVE_SALARY_CHANGE
   FOR EMPLOYEE
   AFTER UPDATE
AS BEGIN
   IF (old.salary <> new.salary) THEN
      INSERT INTO SALARY_HISTORY (EMP_NO, CHANGE_DATE, UPDATER_ID, OLD_SALARY,
                                  PERCENT_CHANGE)
      VALUES (old.emp_no,'now',USER,old.salary,(new.salary-old.salary)*100/old.
salary);
END ^
```

20.6.5 Триггеры, преобразующие неизменяемые представления в изменяемые

Одно из назначений триггеров — преобразование неизменяемых представлений в изменяемые. Примеры таких триггеров см. в главе 15.

Глава 21

Пакеты (PACKAGE)

Пакет — группа процедур и функций, которая представляет собой единый объект базы данных.

Пакеты состоят из двух частей: заголовка (ключевое слово `PACKAGE`) и тела (ключевые слова `PACKAGE BODY`). Сначала создаётся заголовок, а затем — тело.

Пакеты обладают следующими преимуществами:

Модульность

Блоки взаимозависимого кода выделены в логические модули, как это сделано в других языках программирования. В программировании существует множество способов для группировки кода, например с помощью пространств имен (`namespaces`), модулей (`units`) и классов. Со стандартными процедурами и функциями базы данных это невозможно.

Упрощение отслеживания зависимостей

Пакеты упрощают механизм отслеживания зависимостей между набором связанных процедур, а также между этим набором и другими процедурами, как упакованными, так и неупакованными.

Каждый раз, когда упакованная подпрограмма определяет, что используется некоторый объект базы данных, информации о зависимости от этого объекта регистрируется в системных таблицах РЕД Базы Данных. После этого, для того чтобы удалить или изменить этот объект, вы сначала должны удалить, то что зависит от него. Поскольку зависимости от других объектов существуют только для тела пакета, это тело пакета может быть легко удалено, даже если какой-нибудь другой объект зависит от этого пакета. Когда тело удаляется, заголовок остаётся, что позволяет пересоздать это тело после того, как сделаны изменения связанные с удалённым объектом.

Упрощение управления разрешениями

Поскольку РЕД База Данных выполняет подпрограммы с полномочиями вызывающей стороны, то каждой вызывающей подпрограмме необходимо предоставить полномочия на использования ресурсов, если эти ресурсы не являются непосредственно доступными вызывающей стороне. Использование каждой подпрограммы требует предоставления привилегий на её выполнение для пользователей и/или ролей.

У упакованных подпрограмм нет отдельных привилегий. Привилегии действуют на пакет в целом. Привилегии, предоставленные пакетам, действительны для всех подпрограмм тела пакета, в том числе частных, и сохраняются для заголовка пакета.

Частные области видимости

Некоторые процедуры и функции могут быть частными (`private`), а именно их использование разрешено только внутри определения пакета.

Все языки программирования имеют понятие области видимости подпрограмм, которое невозможно без какой-либо формы группировки. Пакеты в этом отношении подобны модулям Delphi. Если подпрограмма не объявлена в заголовке пакета (`interface`), но реализована в теле (`implementation`), то такая подпрограмма становится частной (`private`). Частную подпрограмму возможно вызвать только из её пакета.

21.1 Создание заголовка пакета

Оператор `CREATE PACKAGE` создаёт новый заголовок пакета. Синтаксис оператора представлен в листинге 21.1.

Листинг 21.1. Синтаксис оператора создания заголовка пакета `CREATE PACKAGE`

```
CREATE PACKAGE <имя пакета>
[SQL SECURITY {DEFINER | INVOKER}]
AS
BEGIN
   [ <объявление процедуры> | <объявление функции> ...]
END

<объявление процедуры> ::=
   PROCEDURE <имя процедуры> [(<входной параметр> [, <входной параметр> ...])]
      [RETURNS (<выходной параметр> [, <выходной параметр> ...])]

<объявление функции> ::=
   FUNCTION <имя функции> [(<входной параметр> [, <входной параметр> ...])]
      RETURNS <тип> [COLLATE <сортировка>] [DETERMINISTIC]

<входной параметр> ::= <описание параметра> [{=|DEFAULT} <значение по умолчанию>]

<выходной параметр> ::= <описание параметра>

<описание параметра> ::= <имя параметра> <тип> [NOT NULL]
                         [COLLATE <порядок сортировки>]

<тип> ::= <тип данных SQL>
        | [TYPE OF] <имя домена>
        | TYPE OF COLUMN <имя таблицы/представления>.<имя столбца>

<значение по умолчанию>::= {<литерал> | NULL | <контекстная переменная>}
```

Создать новый заголовок пакета может только администратор и пользователь с привилегией `CREATE PACKAGE`.

Пользователь, создавший заголовок пакета становится владельцем пакета.

Процедуры и функции, объявленные в заголовке пакета, доступны вне тела пакета через полный идентификатор имён процедур и функций (`<имя пакета>.<имя процедуры>` и `<имя пакета>.<имя функции>`). Процедуры и функции, определенные в теле пакета, но не объявленные в заголовке пакета, не видны вне тела пакета.

Имя пакета должно быть уникальным среди имён всех пакетов и может содержать до 63 символов. Имена процедур и функций, объявленные в заголовке пакета, должны быть уникальны среди имён процедур и функций, объявленных в заголовке и теле пакета и могут содержать до 63 символов.

Подробное описание заголовков хранимых процедур и функций можно найти в соответствующих разделах (см. раздел 18.1 и раздел 19.1).

21.1.1 Привилегии выполнения

Необязательное предложение `SQL SECURITY {DEFINER | INVOKER}` определяет, в контексте какого пользователя будет выполняться пакет. Такое поведение действует на пакет в целом и действительны для всех подпрограмм пакета. Ключевое слово `INVOKER` (значение по умолчанию) указывает, что пакет выполняется с правами вызвавшего его пользователя. Задание ключевого слова `DEFINER` означает, что пакет выполняется с правами к объектам базы данных его владельца (создателя). Значение по умолчанию на уровне всей базы данных можно изменить оператором `ALTER DATABASE SET DEFAULT SQL SECURITY`. Для процедур и функций, определенных в пакете, запрещено явно задавать предложение

SQL SECURITY.

21.1.2 Входные параметры

Входные параметры заключаются в скобки после имени хранимой процедуры или функции. Параметры передаются по значению, то есть любые изменения значений входных параметров никак не влияют на значения этих параметров в вызвавшей программе. Входным параметрам может присваиваться значение по умолчанию. Параметры, для которых заданы значения по умолчанию, должны располагаться в самом конце списка. Если входной параметр основан на домене, которому также задано значение по умолчанию в предложении DEFAULT, то новое значение по умолчанию перекрывает указанное при описании домена. Для параметра можно указать ограничение NOT NULL, тем самым запретив передавать в него значение NULL. Для параметра строкового типа существует возможность задать порядок сортировки с помощью предложения COLLATE.

21.1.3 Выходные параметры

Для хранимых процедур список выходных параметров задаётся в необязательное предложение RETURNS.

Для хранимых функций в обязательном предложении RETURNS задаётся тип возвращаемого значения.

21.1.4 Использование доменов при объявлении параметров

Если при описании параметра, локальной переменной указано имя домена, то для него копируются все характеристики этого домена. Если в описании присутствует предложение TYPE OF, то для переменной копируется только тип данных домена, а в случае строковых типов ещё и набор символов, и порядок сортировки.

21.1.5 Использование типа столбца при объявлении параметров

Входные и выходные параметры, а также локальные переменные можно объявлять, используя тип данных столбцов существующих таблиц и представлений. Для этого используется предложение TYPE OF COLUMN, после которого указывается имя таблицы или представления и через точку имя столбца. При использовании TYPE OF COLUMN наследуется только тип данных, а в случае строковых типов ещё и набор символов, и порядок сортировки. Ограничения и значения по умолчанию столбца никогда не используются.

21.1.6 Детерминированные функции

Необязательное предложение DETERMINISTIC в объявлении функции указывает, что функция детерминированная. Детерминированные функции каждый раз возвращают один и тот же результат, если предоставлять им один и тот же набор входных значений. Недетерминированные функции могут возвращать каждый раз разные результаты, даже если предоставлять им один и тот же набор входных значений. Если для функции указано, что она является детерминированной, то такая функция не вычисляется заново, если она уже была вычислена однажды с данным набором входных аргументов, а берет свои значения из кэша метаданных (если они там есть).

21.2 Изменение заголовка пакета

Оператор ALTER PACKAGE изменяет заголовок пакета. Синтаксис оператора представлен в листинге 21.2.

Листинг 21.2. Синтаксис оператора изменения заголовка пакета ALTER PACKAGE

```
ALTER PACKAGE <имя пакета>
[SQL SECURITY {DEFINER | INVOKER}]
AS
BEGIN
   [ <объявление процедуры> | <объявление функции> ...]
END

<объявление процедуры> ::=
   PROCEDURE <имя процедуры> [(<входной параметр> [, <входной параметр> ...])]
      [RETURNS (<выходной параметр> [, <выходной параметр> ...])]

<объявление функции> ::=
   FUNCTION <имя функции> [(<входной параметр> [, <входной параметр> ...])]
      RETURNS <тип> [COLLATE <сортировка>] [DETERMINISTIC]
```

Изменить заголовок пакета может администратор, владелец пакета или пользователь с привилегией ALTER ANY PACKAGE.

Позволяется изменять количество и состав процедур и функций, их входных и выходных параметров. Также данным оператором можно поменять в контексте какого пользователя будет выполняться пакет. При этом исходный код тела пакета сохраняется. Состояние соответствия тела пакета его заголовку отображается в таблице RDB$PACKAGES в столбце RDB$VALID_BODY_FLAG.

Изменить значение SQL SECURITY можно без указания тела пакета:

```
ALTER PACKAGE <имя пакета>
         SQL SECURITY {DEFINER | INVOKER}
       | DROP SQL SECURITY
```

21.3 Создание нового или изменение существующего заголовка пакета

Оператор CREATE OR ALTER PACKAGE создаёт новый или изменяет существующий заголовок пакета. Синтаксис оператора представлен в листинге 21.3.

Листинг 21.3. Синтаксис оператора CREATE OR ALTER PACKAGE

```
CREATE OR ALTER PACKAGE <имя пакета>
[SQL SECURITY {DEFINER | INVOKER}]
AS
BEGIN
   [ <объявление процедуры> | <объявление функции> ...]
END
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
<объявление процедуры> ::=
   PROCEDURE <имя процедуры> [(<входной параметр> [, <входной параметр> ...])]
      [RETURNS (<выходной параметр> [, <выходной параметр> ...])]

<объявление функции> ::=
   FUNCTION <имя функции> [(<входной параметр> [, <входной параметр> ...])]
      RETURNS <тип> [COLLATE <сортировка>] [DETERMINISTIC]
```

Если заголовок пакета не существует, то он будет создан с использованием предложения `CREATE PACKAGE`. Если он уже существует, то он будет изменен и перекомпилирован, при этом существующие привилегии и зависимости сохраняются.

21.4 Удаление заголовка пакета

Оператор `DROP PACKAGE` удаляет существующий заголовок пакета. Синтаксис оператора представлен в листинге 21.4.

Листинг 21.4. Синтаксис оператора удаления заголовка пакета `DROP PACKAGE`

```
DROP PACKAGE <имя пакета>
```

Выполнить удаление заголовка пакета может администратор, владелец пакета или пользователь с привилегией `DROP ANY PACKAGE`.

Перед удалением заголовка пакета, необходимо выполнить удаление тела пакета (`DROP PACKAGE BODY`), иначе будет выдана ошибка. Если от заголовка пакета существуют зависимости, то при попытке удаления такого заголовка будет выдана соответствующая ошибка.

21.5 Создание нового или пересоздание существующего заголовка объекта

Оператор `RECREATE PACKAGE` создаёт новый или пересоздает существующий заголовок пакета. Синтаксис оператора представлен в листинге 21.5.

Листинг 21.5. Синтаксис оператора `RECREATE PACKAGE`

```
RECREATE PACKAGE <имя пакета>
[SQL SECURITY {DEFINER | INVOKER}]
AS
BEGIN
   [ <объявление процедуры> | <объявление функции> ...]
END

<объявление процедуры> ::=
   PROCEDURE <имя процедуры> [(<входной параметр> [, <входной параметр> ...])]
      [RETURNS (<выходной параметр> [, <выходной параметр> ...])]

<объявление функции> ::=
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
   FUNCTION <имя функции> [(<входной параметр> [, <входной параметр> ...])]
      RETURNS <тип> [COLLATE <сортировка>] [DETERMINISTIC]
```

Если заголовок пакета с таким именем уже существует, то оператор попытается удалить его и создать новый заголовок пакета. Пересоздать заголовок пакета невозможно, если у существующей заголовка пакета имеются зависимости или существует тело этого пакета. После пересоздания заголовка пакета привилегии на выполнение подпрограмм пакета и привилегии самого пакета не сохраняются.

21.6 Создание тела пакета

Оператор `CREATE PACKAGE BODY` создаёт новое тело пакета. Тело пакета может быть создано только после того как будет создан заголовок пакета. Если заголовка пакета с именем `<имя пакета>` не существует, то будет выдана соответствующая ошибка. Синтаксис оператора представлен в листинге 21.6.

Листинг 21.6. Синтаксис оператора создания тела пакета `CREATE PACKAGE BODY`

```
CREATE PACKAGE BODY <имя пакета>
AS
BEGIN
   [ <объявление процедуры> | <объявление функции> ...]
   [ <реализация процедуры> | <реализация функции> ...]
END

<объявление процедуры> ::=
   PROCEDURE <имя процедуры> [(<входной параметр> [, <входной параметр> ...])]
      [RETURNS (<выходной параметр> [, <выходной параметр> ...])]

<объявление функции> ::=
   FUNCTION <имя функции> [(<входной параметр> [, <входной параметр> ...])]
      RETURNS <тип> [COLLATE <сортировка>] [DETERMINISTIC]

<реализация процедуры> ::=
   PROCEDURE <имя процедуры> [(<входной_параметр> [, <входной_параметр> ...])]
      [RETURNS (<выходной параметр> [, <выходной параметр> ...])]
      {EXTERNAL NAME '<внешний модуль>' ENGINE <имя движка> [AS <тело вн. проц.>]} |
      { AS
         [<объявление> [<объявление> ...] ]
      BEGIN
         <блок операторов>
      END }

<реализация функции> ::=
   FUNCTION <имя функции> [(<входной_параметр> [, <входной_параметр> ...])]
      RETURNS <тип> [COLLATE <сортировка>] [DETERMINISTIC]
      {EXTERNAL NAME '<внешний модуль>' ENGINE <имя движка> [AS <тело вн. функ.>]} |
      { AS
         [<объявление> [<объявление> ...] ]
      BEGIN
         <блок операторов>
      END }
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
<входной параметр> ::= <описание параметра> [{=|DEFAULT} <значение по умолчанию>]

<выходной параметр> ::= <описание параметра>

<описание параметра> ::= <имя параметра> <тип> [NOT NULL]
                         [COLLATE <порядок сортировки>]

<тип> ::= {
   <тип данных SQL>
 | [TYPE OF] <имя домена>
 | TYPE OF COLUMN <имя таблицы/представления>.<имя столбца> }

<значение по умолчанию>::= {<литерал> | NULL | <контекстная переменная>}

<внешний модуль> ::= '<имя внешнего модуля>!<имя функции в модуле>[!<информация>]'

<объявление> ::= <объявление локальной переменной>;
               | <объявление курсора>;
               | <объявление процедуры/функции>
               | <реализация процедуры/функции>
```

Выполнить оператор создания тела пакета может администратор, владелец пакета или пользователь с привилегией `ALTER ANY PACKAGE`.

Все процедуры и функции, объявленные в заголовке пакета, должны быть реализованы в теле пакета с той же сигнатурой. Кроме того, должны быть реализованы и все процедуры и функции, объявленные в теле пакета, с той же сигнатурой. Процедуры и функции, определенные в теле пакета, но не объявленные в заголовке пакета, не видны вне тела пакета.

Имена процедур и функций, объявленные в теле пакета, должны быть уникальны среди имён процедур и функций, объявленных в заголовке и теле пакета.

Значения по умолчанию для параметров процедур не могут быть переопределены. Это означает, что они могут быть в реализации только для частных процедур, которые не были объявлены.

21.7 Удаление тела пакета

Оператор `DROP PACKAGE BODY` удаляет существующее тело пакета. Синтаксис оператора представлен в листинге 21.7.

Листинг 21.7. Синтаксис оператора удаления тела пакета `DROP PACKAGE BODY`

```
DROP PACKAGE BODY <имя пакета>
```

Выполнить удаление заголовка пакета может администратор, владелец пакета или пользователь с привилегией `DROP ANY PACKAGE`.

21.8 Создание нового или пересоздание существующего тела объекта

Оператор RECREATE PACKAGE BODY создаёт новое или пересоздает существующее тело пакета. Синтаксис оператора представлен в листинге 21.8.

Листинг 21.8. Синтаксис оператора RECREATE PACKAGE BODY

```
RECREATE PACKAGE BODY <имя пакета>
AS
BEGIN
   [ <объявление процедуры> | <объявление функции> ...]
   [ <реализация процедуры> | <реализация функции> ...]
END

<объявление процедуры> ::=
   PROCEDURE <имя процедуры> [(<входной параметр> [, <входной параметр> ...])]
      [RETURNS (<выходной параметр> [, <выходной параметр> ...])]

<объявление функции> ::=
   FUNCTION <имя функции> [(<входной параметр> [, <входной параметр> ...])]
      RETURNS <тип> [COLLATE <сортировка>] [DETERMINISTIC]

<реализация процедуры> ::=
   PROCEDURE <имя процедуры> [(<входной_параметр> [, <входной_параметр> ...])]
      [RETURNS (<выходной параметр> [, <выходной параметр> ...])]
      {EXTERNAL NAME '<внешний модуль>' ENGINE <имя движка> [AS <тело вн. проц.>]} |
      { AS
         [<объявление> [<объявление> ...] ]
      BEGIN
         <блок операторов>
      END }

<реализация функции> ::=
   FUNCTION <имя функции> [(<входной_параметр> [, <входной_параметр> ...])]
      RETURNS <тип> [COLLATE <сортировка>] [DETERMINISTIC]
      {EXTERNAL NAME '<внешний модуль>' ENGINE <имя движка> [AS <тело вн. функ.>]} |
      { AS
         [<объявление> [<объявление> ...] ]
      BEGIN
         <блок операторов>
      END }
```

Если тело пакета с таким именем уже существует, то оператор попытается удалить его и создать новое тело пакета. После пересоздания тела пакета привилегии на выполнение подпрограмм пакета и привилегии самого пакета сохраняются.

Глава 22

Сортировка (COLLATION)

В SQL текстовые строки принадлежат к сортируемым объектам. Это означает, что они подчиняются своим внутренним правилам упорядочения, например, алфавитному порядку. К таким текстовым строкам можно применять операции сравнения (например, "меньше чем" или "больше чем"), при этом значения выражения должны вычисляться согласно определённой последовательности сортировки. Например, выражение 'a'<'b' означает, что 'a' предшествует 'b' в последовательности сортировки. Под выражением 'c'>'b' имеется в виду, что в последовательности сортировки 'c' определено после 'b'. Текстовые строки, включающие больше одного символа, сортируются путём последовательного сравнения символов: сначала сравниваются первые символы двух строк, затем вторые символы и так далее, до тех пор, пока не будет найдено различие между двумя строками. Такое различие управляет порядком сортировки.

Под сортировкой (COLLATION) принято понимать такой объект схемы, который определяет упорядочивающую последовательность (или последовательность сортировки).

22.1 Создание сортировки

Оператор CREATE COLLATION задает новый порядок сортировки для существующего в базе данных набора символов.

Листинг 22.1. Синтаксис оператора CREATE COLLATION

```
CREATE COLLATION <имя сортировки>
FOR <имя набора символов>
[FROM <базовая сортировка> | FROM EXTERNAL ('<имя внешнего файла>')]
[NO PAD | PAD SPACE]
[CASE SENSITIVE | CASE INSENSITIVE]
[ACCENT SENSITIVE | ACCENT INSENSITIVE]
['<атрибут> [; <атрибут> ...]'];

<атрибут> ::= <имя атрибута> = <значение атрибута>
```

Создать новую сортировку может администратор и пользователь с привилегией CREATE COLLATION.

Пользователь, создавший сортировку, становится её владельцем.

Оператор CREATE COLLATION ничего не создаёт, а делает сортировку известной для базы данных. Сортировка уже должна присутствовать в системе, как правило в файле библиотеки, и должна быть зарегистрирована в файле fbintl.conf подкаталога intl корневой директории РЕД Базы Данных.

Необязательное предложение FROM указывает сортировку, на основе которой будет создана новая сортировка. Такая сортировка должна уже присутствовать в базе данных. Если указано ключевое слово EXTERNAL, то будет осуществлён поиск сортировки из файла intl/fbintl.conf, при этом имя внешнего файла должно в точности соответствовать имени в конфигурационном файле (чувствительно к регистру).

Если предложение FROM отсутствует, то система ищет в конфигурационном файле fbintl.conf подкаталога intl корневой директории сервера сортировку с именем, указанным сразу после CREATE COLLATION.

Если указана опция NO PAD, то в сортировке конечные пробелы при сравнении учитываются. Если указана опция PAD SPACE, то конечные пробелы при сравнении не учитываются.

Необязательное предложение CASE позволяет указать будет ли сравнение чувствительно к регистру.

Необязательное предложение ACCENT позволяет указать будет ли сравнение чувствительно к акцентированным буквам (например "e" и "ё").

В операторе CREATE COLLATION можно также указать атрибуты для сортировки. Ниже в таблице приведён список доступных атрибутов. Не все атрибуты применимы ко всем сортировкам. Если атрибут не применим к сортировке, но указан при её создании, то это не вызовет ошибки. Имена атрибутов чувствительны к регистру.

Таблица 22.1 — Список доступных атрибутов COLLATION

Имя	Значение	Валидность	Описание
DISABLE-COMPRESSIONS	0, 1	1 bpc	Отключение сжатия. Сжатия заставляют определённые символьные последовательности быть сортированными как атомарные модули, например, испанские c + h как единственный символ ch.
DISABLE-EXPANSIONS	0, 1	1 bpc	Отключение расширений. Расширения позволяют рассматривать определённые символы (например, лигатуры или гласные умляуты) как последовательности символов и соответственно сортировать.
ICU-VERSION	default или M.N	UNI	Задаёт версию библиотеки ICU для использования. Допустимые значения определены в соответствующих элементах <intl_module> в файле intl/fbintl.conf. Формат: либо строка default или основной и дополнительный номер версии, как 3.0.
LOCALE	xx_YY	UNI	Задаёт параметры сортировки языкового стандарта. Требуется полная версия библиотеки ICU. Формат строки: du_NL.
MULTI-LEVEL	0, 1	1 bpc	Использование нескольких уровней сортировки.
NUMERIC-SORT	0, 1	UNI	Обрабатывает непрерывные группы десятичных цифр в строке как атомарные модули и сортирует их в числовой последовательности (известна как естественная сортировка).
SPECIALS-FIRST	0, 1	1 bpc	Сортирует специальные символы (пробелы и т.д.) до буквенно-цифровых символов.

"1 bpc" в таблице указывает на то, что атрибут действителен для сортировок наборов символов, использующих 1 байт на символ, а "UNI" — для юникодных сортировок.

Пример.

Создание регистронезависимой сортировки на основе уже присутствующей в базе данных со специфичными атрибутами.

```
CREATE COLLATION ES_ES_CI_COMPR
FOR ISO8859_1
FROM ES_ES
CASE INSENSITIVE
'DISABLE-COMPRESSIONS=0';
```

22.2 Удаление сортировки

Оператор DROP COLLATION удаляет указанную сортировку. Сортировка должна присутствовать в базе данных, иначе будет выдана соответствующая ошибка.

Листинг 22.2. Синтаксис оператора DROP COLLATION

```
DROP COLLATION <имя сортировки>;
```

Выполнить данный оператор может только администратор, владелец сортировки и пользователь с привилегией DROP ANY COLLATION.

Глава 23

Планировщик заданий

В СУБД РЕД База Данных реализован встроенный планировщик заданий для выполнения плановых работ. Планировщик имеет возможность запускать задания по расписанию (с указанием интервалов времени и дней недели) и оповещать о запуске и завершении заданий, а также об ошибках. В качестве «задания» может выступать блок PSQL операторов или команды операционной системы.

На данный момент планировщик заданий не работает на архитектуре `Classic` в `Linux`.

Задания сохраняются в базе данных `scheduler.fdb` в таблице `SCH$SCHEDULED_JOBS`. Путь к базе задается в файле конфигурации `scheduler.conf` в параметре `SchedulerDatabase`.

Таблица 23.1 — Описание полей таблицы `SCH$SCHEDULED_JOBS`

Поле	Тип	Описание
`SCH$JOB_NAME`	`CHAR(63)`	Имя задания
`SCH$JOB_ID`	`INTEGER`	ID задания
`SCH$JOB_SOURCE`	`BLOB TEXT`	Код задания в текстовом виде
`SCH$JOB_BLR`	`BLOB BINARY`	Скомпилированный BLR для задания
`SCH$DESCRIPTION`	`BLOB TEXT`	Содержит описание задания
`SCH$OWNER_NAME`	`CHAR(63)`	Имя пользователя, создавшего задание
`SCH$JOB_INACTIVE`	`SMALLINT`	Имеет два значения: 1 - задание не будет запускаться по расписанию 0 - задание будет запускаться
`SCH$JOB_TYPE`	`SMALLINT`	Имеет два значения: 0 - обычное PSQL задание 1 - задание с командой ОС
`SCH$JOB_SCHEDULE`	`VARBINARY(64)`	Строка в формате `cron`, задающая расписание
`SCH$START_DATE`	`TIMESTAMP`	Дата и время начала выполнения задания
`SCH$END_DATE`	`TIMESTAMP`	Дата и время окончания выполнения задания
`SCH$DATABASE`	`VARCHAR(255)`	Имя базы данных, для которой создано задание

События, связанные с заданиями, регистрируются в таблице `SCH$JOBS_LOG` (`scheduler.fdb`).

Таблица 23.2 — Описание полей таблицы `SCH$JOBS_LOG`

Поле	Тип	Описание
`SCH$TIMESTAMP`	`TIMESTAMP`	Время произошедшего события
`SCH$JOB_NAME`	`CHAR(63)`	Имя задания
`SCH$JOB_ID`	`INTEGER`	ID задания
`SCH$EVENT`	`VARCHAR(32)`	Регистрируются следующие типы событий: `RUN_START` - начало выполнения задания; `RUN_FINISH` - завершение выполнения задания; `RUN_ERROR` - ошибка во время выполнения задания.

(разрыв таблицы)

(разрыв таблицы)

Поле	Тип	Описание
SCH$MESSAGE	VARCHAR(1023)	Текст ошибки

Так как к базе данных scheduler.fdb не имеет права подключаться никто, то для всех подключений доступны соответствующие виртуальные системные таблицы RDB$JOBS и RDB$JOBS_LOG (аналогичные таблицам SCH$SCHEDULED_JOBS и SCH$JOBS_LOG). В этих таблицах обычные пользователи видят информацию только о тех заданиях, которые они создали для любой базы данных; SYSDBA видит все задания для всех баз данных.

Каждое задание выполняется в отдельном подключении от имени пользователя, который его создал. Если необходимо прервать выполнение задания, то нужно удалить соответствующую строку из системной таблицы MON$ATTACHMENTS. Сведения об имени выполняемого задания можно увидеть в поле MON$CLIENT_VERSION.

Критические ошибки, при которых работа планировщика невозможна (например, отсутствие или повреждение базы данных scheduler.fdb), записываются в firebird.log.

23.1 Создание задания

Оператор создания задания становится доступным после подключения к базе данных с правами администратора или привилегией CREATE JOB.

Листинг 23.1. Синтаксис оператора создания задания CREATE JOB

```
CREATE JOB <имя_задания>
<параметры расписания>
[ACTIVE | INACTIVE]
[START DATE '<timestamp>' | NULL]
[END DATE '<timestamp>' | NULL]
{AS
    <объявления переменных>
BEGIN
    <блок sql-операторов>
END
| COMMAND '<команда>' }

 <параметры расписания> ::= '<Минуты> <Часы> <Дни_месяца> <Месяцы> <Дни_недели>'
```

Имя задания должно быть уникальным среди всех баз данных.

Ключевое слово ACTIVE (по умолчанию) указывает, что задание будет запускаться по расписанию. Ключевое слово INACTIVE означает, что задание запускаться не будет.

«Параметры расписания» задаются в виде строки в формате, аналогичном cron.

Необязательные предложения START DATE и END DATE задают промежуток времени, в котором будет запускаться задание, в формате 'ДД.ММ.ГГГГ ЧЧ:ММ'. Указанные значения включаются в интервал выполнения. Значение NULL используется, чтобы убрать интервал.

Предложение COMMAND дает возможность запланировать выполнение команды ОС. Такой тип заданий может создавать только SYSDBA. Чтобы задать список программ, которые можно использовать в команде, необходимо настроить параметр JobCommandAccess в scheduler.conf. В значении параметра можно указывать как пути к конкретным программам, так и директории с программами. По умолчанию разрешено использовать только программы из директории СУБД.

Задание выполняется от имени пользователя, который его создал.

23.2 Изменение задания

Синтаксис изменения задания аналогичен его созданию:

Листинг 23.2. Синтаксис оператора изменения задания ALTER JOB

```
ALTER JOB <имя_задания>
[<параметры расписания>]
[ACTIVE | INACTIVE]
[START DATE '<timestamp>' | NULL]
[END DATE '<timestamp>' | NULL]
[AS
    <объявления переменных>
BEGIN
    <блок sql-операторов>
END
| COMMAND '<команда>' ]

<параметры расписания> ::= '<Минуты> <Часы> <Дни_месяца> <Месяцы> <Дни_недели>'
```

Изменить задание может его создатель и SYSDBA.

23.3 Удаление задания

Оператор удаления задания доступен создателю задания и SYSDBA.

Листинг 23.3. Синтаксис оператора удаления задания DROP JOB

```
DROP JOB <имя_задания>;
```

23.4 Оповещения

Для настройки оповещений о статусе заданий создан параметр JobNotificationCommand в scheduler.conf, в котором можно указать команду ОС. В команде можно использовать следующие переменные со сведениями о произошедшем событии:

- $(timestamp) – дата и время события;
- $(job_name) – имя задания;
- $(job_id) – ID задания;
- $(event) – тип события: RUN_START, RUN_FINISH или RUN_ERROR;
- $(message) – сообщение с описанием ошибки.

Linux:

```
echo $(timestamp) $(job_name) $(job_id) $(event) $(message) >> /home/user/
notifications.txt
```

Windows:

```
echo $(timestamp) $(job_name) $(job_id) $(event) $(message) >> C:\RDB\notifications.
txt
```

23.5 Примечание для задания

Для существующего задания можно добавить комментарий, используя оператор `COMMENT ON`:

Листинг 23.4.

```
COMMENT ON
   JOB <имя задания> IS {'<текст примечания>' | NULL};
```

Значение `NULL` удаляет существующее примечание.

Выполнить оператор `COMMENT ON JOB` могут администраторы, владельцы домена или пользователи с привилегией `ALTER ANY JOB`.

Глава 24

Внешние хранимые процедуры, функции и триггеры, написанные на языке Java

В РЕД Базе Данных реализована возможность создания внешних процедур, функций и триггеров с использованием языка программирования Java, что существенно расширяет возможности языка PSQL по обработке данных. Например, с помощью PSQL невозможно работать с объектами файловой системы, а язык Java позволяет это.

О настройках РЕД Базы Данных для работы с внешними подпрограммами и о взаимодействиях с Java методами из базы данных описано в отдельном руководстве "Внешние хранимые процедуры, функции и триггеры, написанные на языке Java". В этом же разделе рассмотрен SQL синтаксис операторов создания, изменения и пересоздания внешних хранимых подпрограмм, написанных на Java.

24.1 Объявление/изменение/пересоздание внешних процедур

Синтаксис операторов создания, изменения и пересоздания внешней хранимой процедуры, написанной на Java, имеет одинаковую структуру и приведен в листинге:

Листинг 24.1. Синтаксис операторов создания, изменения и пересоздания внешней хранимой процедуры

```
{CREATE [ OR ALTER ] | RECREATE | ALTER} PROCEDURE <имя хранимой процедуры>
[AUTHID {OWNER | CALLER}]
   [(<входной параметр> [, <входной параметр> ...])]
[RETURNS (<выходной параметр> [, <выходной параметр> ...])]
[SQL SECURITY {DEFINER | INVOKER}]
EXTERNAL NAME '<полное имя класса>.<имя static метода>!(
              [<Java тип> [, <Java тип>...]])'
              [!<определяемая пользователем информация>]
ENGINE JAVA
```

Внешние процедуры либо не имеют выходных параметров, либо возвращают набор данных `ExternalResultSet` (или класс, реализующий этот интерфейс), в зависимости от того, является ли процедура селективной или нет. Выходные параметры при вызове функций должны быть массивами. JavaEngine передает каждый параметр как массив с длиной 1, таким образом процедуры могут менять их нулевой элемент.

Пример 1.

Пример объявления в базе данных внешней выполнимой процедуры, осуществляющей вставку записи в таблицу:

```
CREATE PROCEDURE  testInsert (n integer, s varchar(10))
EXTERNAL NAME 'example.ExampleClass.insert(int, String)'
ENGINE JAVA;
```

В следующем листинге приведено описание внешней процедуры, написанной на Java, иллюстрирующей пример вставки записи в таблицу:

```
public static void insert(int n, String s) throws SQLException {
   Connection con = DriverManager.getConnection("jdbc:default:connection:");
   try {
         PreparedStatement stmt = con.prepareStatement ("insert into test_table (n, s)
                                                          values (?, ?)");
         try { stmt.setInt(1, n);
               stmt.setString(2, s);
               stmt.execute(); }
         finally { stmt.close(); }
   }
   finally { con.close(); }
}
```

Пример 2.

Пример объявления в базе данных внешней селективной процедуры, генерирующей строки:

```
CREATE PROCEDURE  testGenRows (numRows integer) RETURNS (n integer)
EXTERNAL NAME 'example.ExampleClass.genRows(int, int[])'
ENGINE JAVA;
```

Пример тела внешней процедуры, осуществляющей генерирование значений строк, приведен ниже:

```
public static ExternalResultSet genRows(final int numRows, final int[] n) {
   return new ExternalResultSet() {
      private int i = 1;
      public void close() {}
      public boolean fetch() throws Exception {
         if (i > numRows) {
            return false;
         }
         n[0] = i++;
         return true;
      }
   };
}
```

24.2 Объявление/изменение/пересоздание внешних функций

Синтаксис операторов создания, изменения и пересоздания внешней функции, написанной на Java, имеет одинаковую семантику и приведен в листинге:

Листинг 24.2. Синтаксис операторов создания, изменения и пересоздания внешней функции

```
{CREATE [ OR ALTER ] | RECREATE | ALTER} FUNCTION <имя функции>
[AUTHID {OWNER | CALLER}]
   [(<входной параметр> [, <входной параметр> ...])]
RETURNS (<тип данных>)
[SQL SECURITY {DEFINER | INVOKER}]
EXTERNAL NAME '<полное имя класса>.<имя static метода>!(
              [<Java тип> [, <Java тип>...]])'
              [!<определяемая пользователем информация>]
ENGINE JAVA
```

В отличие от внешних процедур, внешние функции всегда возвращают одно значение какого-либо из типов описанных в разделе "Соответствие типов данных SQL и Java" руководства разработчика.

Пример 1.

Пример объявления новой или изменения существующей внешней функции, возвращающей системное свойство по указанному ключу:

```
CREATE OR ALTER FUNCTION get_system_property (name varchar(80))
RETURNS varchar(80)
EXTERNAL NAME 'java.lang.System.getProperty(String)'
ENGINE JAVA;
```

Пример 2.

Описанная в примере внешняя функция производит суммирование входных параметров функции, объявленной в базе данных:

```
public static int sum() throws SQLException {
   FunctionContext context = FunctionContext.get();
   ValuesMetadata valuesmetadata = context.getInputMetadata();
   Values values = context.getInputValues();
   int ret = 0;
   for (int i = 1; i <= valuesmetadata.getParameterCount(); ++i) {
      ret += ((BigDecimal) values.getObject(i)).intValue();
   }
   return ret;
}
```

Пример регистрации в базе данных этой внешней функции:

```
CREATE FUNCTION funcSum2 (n1 integer, n2 integer)
RETURNS integer
EXTERNAL NAME 'example.ExampleClass.sum()'
ENGINE JAVA;
```

24.3 Объявление/изменение/пересоздание внешних триггеров

Синтаксис операторов создания, изменения и пересоздания внешнего триггера, написанного на Java, имеет одинаковую семантику и приведен в листинге:

Листинг 24.3. Синтаксис операторов создания, изменения и пересоздания внешнего триггера

```
{CREATE [ OR ALTER ] | RECREATE | ALTER} TRIGGER <имя триггера>
[AUTHID {OWNER | CALLER}]
{
   <объявление табличного триггера>
 | <объявление табличного триггера в стандарте SQL-2003>
 | <объявление триггера базы данных>
 | <объявление DDL триггера> }
[SQL SECURITY {DEFINER | INVOKER}]
EXTERNAL NAME '<полное имя класса>.<имя static метода>!(
              [<Java тип> [, <Java тип>...]])'
              [!<определяемая пользователем информация>]
ENGINE JAVA
```

Спецификация вызовов триггеров всегда без параметров, и Java метод должен возвращать `void`. Детали вызова и значения полей `OLD` и `NEW` можно получить и установить из контекста вызова.

Пример

Тело внешнего триггера, написанного на Java, выполняющего логгирование:

```
public static void info() throws SQLException {
   Logger log = LoggerFactory.getLogger(ExampleClass.class);
   String NEWLINE = System.getProperty("line.separator");
   TriggerContext context = TriggerContext.get();
   String msg = "Table: "+ context.getTableName() +
                "; Type: "+ context.getType() +
                "; Action: "+ context.getAction() +
                 valuesToStr(context.getFieldsMetadata(), context.getOldValues(),
                             NEWLINE + "OLD:"+ NEWLINE) +
                 valuesToStr(context.getFieldsMetadata(), context.getNewValues(),
                             NEWLINE + "NEW:"+ NEWLINE);
   log.info(msg);
}

private static String valuesToStr(ValuesMetadata metadata, Values values, String
label) throws SQLException {
   if (values == null)
      return "";
   StringBuilder sb = new StringBuilder(label);
   String NEWLINE = System.getProperty("line.separator");
   for (int i = 1, count = metadata.getParameterCount(); i <= count; ++i)
      sb.append(metadata.getName(i) + ": "+ values.getObject(i) + NEWLINE);
   return sb.toString();
}
```

Пример объявления нового или изменения существующего внешнего триггера в базе данных:

```
CREATE OR ALTER TRIGGER trig_Employee_Log
BEFORE DELETE OR INSERT OR UPDATE on employee
EXTERNAL NAME 'example.ExampleClass.info()'
ENGINE JAVA;
```

24.4 Удаление внешних процедур, функций и триггеров

Удаление внешней процедуры осуществляется оператором `DROP PROCEDURE`.

Листинг 24.4. Синтаксис оператора удаления процедуры

```
DROP PROCEDURE <имя_процедуры>;
```

В отличие от обычных хранимых процедур (функций), сервер не может проконтролировать наличие вызовов удаляемой процедуры (функции) из других внешних процедур, функций или триггеров. Поэтому удаление такой процедуры (функции) пройдет без ошибок, однако при последующем вызове внешней процедуры, функции или триггера будет сгенерировано исключение.

Удаление внешней функции осуществляется оператором `DROP FUNCTION`. Его синтаксис:

Листинг 24.5. Синтаксис оператора удаления функции

```
DROP FUNCTION <имя_функции>;
```

Оператор `DROP TRIGGER` удаляет существующий триггер:

Листинг 24.6. Синтаксис оператора удаления триггера

```
DROP TRIGGER <имя_триггера>;
```

24.5 Вызов внешних процедур и функций

Вызов внешних процедур и функций, написанных на Java, аналогичен вызову обычных хранимых процедур и функций. Например, вызов внешней процедуры можно осуществить оператором `EXECUTE PROCEDURE`. При этом внешняя процедура всегда будет выполняться с правами вызывающего её пользователя.

Глава 25

Полнотекстовый поиск

Полнотекстовый поиск доступен только в промышленной редакции СУБД РЕД База Данных. Подробнее различия функционала редакций СУБД РЕД База Данных описаны в руководстве администратора в разделе "Редакции СУБД РЕД База Данных 5.0".

В «РЕД База Данных» реализованы средства полнотекстового поиска, которые позволяют очень быстро находить нужную информацию в больших объемах текста. Такой тип поиска предусматривает создание соответствующего полнотекстового индекса. Индекс — это объект, создаваемый системой `Lucene` на основе анализа содержимого документа, необходимый для организации поиска. Он представляет собой своеобразный словарь упоминаний слов в полях.

Полнотекстовый поиск основан на свободно распространяемой библиотеке `Lucene`. Эта библиотека предоставляет функции индексирования и поиска, доступ к этим функциям реализуется через API Lucene.

Полнотекстовый поиск позволяет:

- производить поиск по неточному соответствию — искать требуемую информацию при наличии орфографических ошибок в документе или в запросе;
- производить морфологический поиск — поиск с учётом морфологии (всех возможных форм слова);
- производить поиск по нескольким объектам БД (поиск по нескольким столбцам и таблицам).

Поиск может осуществляться по текстовым полям (`CHAR, VARCHAR`), а также по бинарным и текстовым `BLOB`-полям. При этом `BLOB`-поля бинарного типа могут содержать внутри себя документы следующих приложений:

- Acrobat (pdf);
- MS Word (doc);
- MS Excel (xls);
- Microsoft PowerPoint;
- RTF;
- Open Office Writer (odt);
- Html.

В таблице, для которой применяется полнотекстовый поиск, наличие хотя бы одного первичного ключа обязательно.

25.1 Миграция полнотекстового поиска с версии 3.0 на 5.0

После успешной миграции базы данных (подробнее о миграции базы данных см. Руководство администратора глава "Миграции с версии 3.0") для работы полнотекстового поиска необходимо выполнить следующие действия:

1. В конфигурационном файле `plugins.conf` добавить плагин `FTS`:

```
Plugin = FTS {
    Module = $(dir_plugins)/fts
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
    Config = FTS_config
}

Config = FTS_config {
    JarDirs = $(this)/jar/fts
}
```

2. Начиная с РЕД Базы Данных 5 полнотекстовый поиск работает с помощью плагина репликации. В конфигурационном файле `replication.conf` нужно включить плагин `FTS` для базы данных с полнотекстовым поиском. Можно указать отдельную базу:

```
database = /db/employee.fdb
{
    plugin = FTS
}
```

   Или включить для всех баз:

```
database
{
    plugin = FTS
}
```

3. Последовательно выполнить скрипты обновления `FTS` из каталога `misc/upgrade/fts`.
4. Запустить процедуру создания полнотекстовых индексов `FTS$FULL_REINDEX`.

25.2 Настройка сервера для работы полнотекстового поиска

В РЕД Базе Данных в системе полнотекстового поиска используется реализация `Lucene` на языке Java, поэтому для использования описываемого в этом руководстве функционала необходимо установить JRE не ниже 11.

1. Настройте параметры взаимодействия сервера «РЕД База Данных» с виртуальной машиной Java с помощью конфигурационного файла `plugins.conf`, который расположен в корневом каталоге установки сервера. В нем необходимо раскомментировать секции `Plugin=JAVA` и `Config=JAVA_config`, а также `Plugin = FTS` и `Config = FTS_config`. Если они отсутствуют, то добавить:

```
Plugin = FTS {
  Module = $(dir_plugins)/fts
  Config = FTS_config
}

Config = FTS_config {
  JarDirs = $(this)/jar/fts
}
```

2. В `firebird.conf` установите путь к JDK в `JavaHome`, например:

```
JavaHome = /usr/lib/jvm/java-11-openjdk-amd64
```

3. Подключитесь к базе данных безопасности `java-security.fdb` и назначьте права доступа пользо-

вателям базы данных, использующим код Java (подробнее см. раздел 25.2.1). Для этого необходимо пролить скрипт `fts_permissions.sql`, который находится в папке `misc` корневом каталоге сервера.

```
isql -u sysdba -p masterkey -i /opt/RedDatabase/misc/fts_permissions.sql
localhost:/opt/RedDatabase/java-security.fdb
```

В этом скрипте добавляются разрешения для роли `FTS`, поэтому при работе с полнотекстовым поиском пользователь должен подключаться с ролью `FTS`.

Чтобы назначить права на выполнение любого java-кода в `java-security.fdb` вставьте такую запись:

```
insert into permission values(1,'java.security.AllPermission',null, null);
```

Механизм `java-security` устарел и будет удален в версии 6.0.

4. В файле конфигурации `firebird.conf` или `databases.conf` укажите пути к Java-библиотекам, которые реализуют функции полнотекстового поиска:

```
Classpath = ["$(jar)/fts/*.jar"]
```

Значения указанные в `Classpath` в `databases.conf` добавляются к значениям указанным в `firebird.conf`. Если список jar для какой-либо базы данных нужно полностью заменить, то нужно использовать опцию `OverrideClasspath` в `databases.conf`.

5. Если для БД необходим отдельный каталог для хранения индексов, тогда в файле конфигурации `databases.conf` задайте параметр `FTSDirectory` с необходимым каталогом:

```
database.fdb = /path/to/database.fdb
{
  FTSDirectory = /path/to/custom/fts
}
```

6. Начиная с версии Java 18 в `jvm.args` нужно использовать настройку:

```
-Djava.security.manager=allow
```

Установка этого свойства позволяет разрешить динамическую установку Security Manager. Это необходимо, так как начиная с Java 18 по умолчанию это свойство имеет значение `disallow`, и JavaEngine может столкнуться с ошибкой.

7. В `jvm.args` установите директорию для хранения индекса (по умолчанию `/tmp/RDBLuceneIndex/`)

```
-Dfts.directory=...
```

8. По умолчанию в `fts.directory` для БД будет создан каталог (если в `databases.conf` для БД не задан параметр `FTSDirectory`), имя которого эквивалентно GUID БД, в котором будут храниться индексы. Если необходимо отключить создание каталога (хранить все индексы в `fts.directory`), установите параметр `fts.disableGUID`:

```
-Dfts.disableGUID=true
```

9. Также в `jvm.args` можно указать параметры для установки максимального размера индексируемого документа `fts.max_document_size` (по умолчанию установлен максимальный размер `2147483647`) и пропуска битых документов `fts.skip_corrupted` (по умолчанию выключен), на-

пример:

```
-Dfts.max_document_size=10000
-Dfts.skip_corrupted=true
```

10. Для включения многопоточности при переиндексации задайте необходимое количество потоков параметром `fts.thread_count` в `jvm.args`. Допустимы значения от 1 до 64. Если параметр не указан, то по умолчанию используется 1 поток.

25.2.1 Безопасность

Одной из наиболее важных особенностей платформы Java является система безопасности, так называемая «песочница». JavaEngine интегрирует механизм безопасности J2SE/JAAS с РЕД Базой Данных, так что права могут быть назначены пользователям базы данных, использующим код Java.

Права доступа пользователей действуют на уровне сервера. Они хранятся в базе данных безопасности `java-security.fdb`. Эта база содержит следующие таблицы:

Таблицы	Поля
PERMISSION_GROUP	ID, NAME
PERMISSION	PERMISSION_GROUP, CLASS_NAME, ARG1, ARG2
PERMISSION_GROUP_GRANT	PERMISSION_GROUP, DATABASE_PATTERN, GRANTEE_TYPE, GRANTEE_PATTERN

Таблицы `PERMISSION_GROUP_GRANT` и `PERMISSION` содержат внешний ключ, ссылающийся на столбец `ID` таблицы `PERMISSION_GROUP`.

В таблице `PERMISSION` есть столбец `CLASS_NAME`, в котором хранится имя Java класса, предоставляющего доступ к системным ресурсам (см. `java.security.Permission`), и столбцы `ARG1/ARG2`, в котором хранятся аргументы, переданные конструктору этого класса.

Таблица `PERMISSION_GROUP_GRANT` связывает `PERMISSION_GROUP` с пользователями и ролями РЕД Базы Данных. Эта связь осуществляется с помощью `DATABASE_PATTERN` и `GRANTEE_TYPE/GRANTEE_PATTERN`. Шаблоны имеют синтаксис оператора `SIMILAR TO` с символом экранирования `'&'`. `GRANTEE_TYPE` определяет, относится ли `GRANTEE_PATTERN` к `ROLE` или `USER`.

База данных `java-security.fdb` изначально не пустая, в ней создана группа `COMMON` с некоторыми полномочиями для всех пользователей всех баз данных (шаблон `'%'`).

Таблица 25.2 — Предоставленные по умолчанию права

CLASS_NAME	ARG1	ARG2
java.util.PropertyPermission	file.separator	read
java.util.PropertyPermission	java.version	read
java.util.PropertyPermission	java.vendor	read
java.util.PropertyPermission	java.vendor.url	read
java.util.PropertyPermission	line.separator	read
java.util.PropertyPermission	os.*	read
java.util.PropertyPermission	path.separator	read
java.util.PropertyPermission	jna.encoding	read
java.util.PropertyPermission	jna.profiler.prefix	read

25.3 Служебные объекты для полнотекстового поиска

Для функционирования полнотекстового поиска в базе данных должны присутствовать все необходимые служебные объекты. Для создания этих объектов можно выполнить скрипт инициализации `fts.sql`, который находится в папке `misc` корневом каталоге сервера.

```
isql -u sysdba -p masterkey -i /opt/RedDatabase/misc/fts.sql localhost:/tmp/test_fts.
fdb
```

25.3.1 Служебные домены

В полнотекстовом поиске используются три служебных домена.

Таблица 25.3 — Служебные домены

Домен	Тип	Описание
FTS$OBJECT_NAME	VARCHAR(31) CHARACTER SET UNICODE_FSS	Используется для указания имен объектов (индексов, таблиц, полей, триггеров).
FTS$NAME	VARCHAR(255) CHARACTER SET UNICODE_FSS	Используется для указания параметров полнотекстового поиска.
FTS$ROW_ID	CHAR(8) CHARACTER SET OCTETS	Используется для объявления полей, в которых будет использоваться RDB$DB_KEY[1].
FTS$KEY	BLOB	Используется для объявления полей, в которых будет храниться ключ полнотекстового индекса для записи.

25.3.2 Служебные таблицы

В полнотекстовом поиске используются три служебные таблицы.

FTS$INDICES

В служебной таблице FTS$INDICES хранятся метаданные индексов.

Таблица 25.4 — Структура таблицы FTS$INDICES

Имя поля	Тип	Описание
FTS$INDEX_NAME	VARCHAR(31) CHARACTER SET UNICODE_FSS	Имя индекса.
FTS$DESCRIPTION	BLOB SUB_TYPE 1 CHARACTER SET UNICODE_FSS	Комментарий к индексу.

(разрыв таблицы)

[1] RDB$DB_KEY – это "номер записи". Его можно использовать в качестве уникального идентификатора записи, так же как и ее поле первичного ключа. Однако rdb$db_key по ходу работы может меняться. Физически он представляет собой номер таблицы, номер страницы и смещение на запись (причем не на конкретную версию, а вообще на пакет версий этой записи, если они есть).

(разрыв таблицы)

Имя поля	Тип	Описание
FTS$ANALYZER	VARCHAR(255) CHARACTER SET UNICODE_FSS	Служебная информация, имя анализатора.
FTS$INDEX_STATUS	CHAR(1) CHARACTER SET UNICODE_FSS	Статус индекса. Это поле может принимать следующие значения: 'I' inactive – индекс неактивный; 'N' new – индекс создан, требует полное переиндексирование; 'U' needs metadata update – требуется изменение метаданных, триггеров и т.д.; 'D' drop – индекс отмечен к удалению; 'C' complete – для индекса сделаны все изменения в метаданных и он индексируется.
FTS$OCR_DISABLE	BOOLEAN	Статус включения оптического распознавания изображений. Если значение равно false, то для индекса будет применяться оптическое распознавание изображений. При значении true распознавание в изображениях будет отключено.
FTS$OCR_LANGUAGE	FTS$NAME	Служебная информация. Устанавливает язык для оптического распознавания изображений.

FTS$INDEX_SEGMENTS

В таблице FTS$INDEX_SEGMENTS хранятся данные о составе (сегментах) индексов — метаданные полей, входящих в индекс.

Таблица 25.5 — Структура таблицы FTS$INDEX_SEGMENTS

Имя поля	Тип	Описание
FTS$INDEX_NAME	VARCHAR(31) CHARACTER SET UNICODE_FSS	Имя индекса.
FTS$RELATION_NAME	VARCHAR(31) CHARACTER SET UNICODE_FSS	Индексируемая таблица.
FTS$FIELD_NAME	VARCHAR(31) CHARACTER SET UNICODE_FSS	Индексируемое поле.

FTS$POOL

Таблица FTS$POOL содержит значения ключей полнотекстового индекса FTS$KEY для измененных, но не проиндексированных полей.

Таблица 25.6 — Структура таблицы FTS$POOL

Имя поля	Тип	Описание
FTS$KEY	BLOB	Хранит значение ключа для полнотекстового индекса.

(разрыв таблицы)

(разрыв таблицы)

Имя поля	Тип	Описание
FTS$STATUS	SMALLINT	Отображает статус записи. Может принимать три значения: • 1 - запись добавлена; • 2 - запись изменена; • 3 - запись удалена.

25.3.3 Служебные хранимые процедуры

FTS$CREATE_INDEX

Процедура FTS$CREATE_INDEX используется для создания индекса.

Таблица 25.7 — Входные параметры процедуры FTS$CREATE_INDEX

Имя поля	Тип	Описание
FTS$INDEX_NAME	VARCHAR(31) CHARACTER SET UNICODE_FSS	Имя индекса.
FTS$ANALYZER	VARCHAR(255) CHARACTER SET UNICODE_FSS	Имя анализатора. Значение по умолчанию Standard[2].
FTS$DESCRIPTION	BLOB	Комментарии к индексу. Значение по умолчанию NULL.
FTS$OCR_DISABLE	BOOLEAN	Отключает оптическое распознавание изображений. Значение по умолчанию false.
FTS$OCR_LANGUAGE	FTS$NAME	Устанавливает языки для оптического распознавания изображений. Значение по умолчанию eng+rus.

Значение входного параметра FTS$ANALYZER определяет, какой тип анализатора будет использоваться при индексации добавляемого поля:

Таблица 25.8 — Соответствие имен анализаторов и языков

Имя анализатора	Язык
English	Английский
Standard	Английский
Russian	Русский
German	Немецкий
French	Французский
Czech	Чешский
Brazilian	Бразильский
Chinese	Китайский
Dutch	Голландский

(разрыв таблицы)

[2] Допустимые значения имен анализаторов приведены в таблице 25.8.

(разрыв таблицы)

Имя анализатора	Язык
Greek	Греческий
CJK	Китайское письмо

Корректность результатов поиска напрямую зависит от типа выбранного анализатора

FTS$DROP_INDEX

Для удаления индекса из системы полнотекстового поиска используется процедура FTS$DROP_INDEX.

Таблица 25.9 — Входные параметры процедуры FTS$DROP_INDEX

Имя	Тип	Описание
FTS$INDEX_NAME	VARCHAR(31) CHARACTER SET UNICODE_FSS	Имя индекса.

FTS$ADD_FIELD_TO_INDEX

Процедура FTS$ADD_FIELD_TO_INDEX добавляет индексируемое поле в индекс.

Таблица 25.10 — Входные параметры процедуры FTS$ADD_FIELD_TO_INDEX

Имя	Тип	Описание
FTS$INDEX_NAME	VARCHAR(31) CHARACTER SET UNICODE_FSS	Имя индекса.
FTS$RELATION_NAME	VARCHAR(31) CHARACTER SET UNICODE_FSS	Индексируемая таблица.
FTS$FIELD_NAME	VARCHAR(31) CHARACTER SET UNICODE_FSS	Индексируемое поле.

FTS$DROP_FIELD_FROM_INDEX

Процедура FTS$DROP_FIELD_FROM_INDEX используется для удаления индексируемого поля из индекса.

Таблица 25.11 — Входные параметры процедуры FTS$DROP_FIELD_FROM_INDEX

Имя	Тип	Описание
FTS$INDEX_NAME	VARCHAR(31) CHARACTER SET UNICODE_FSS	Имя индекса.
FTS$RELATION_NAME	VARCHAR(31) CHARACTER SET UNICODE_FSS	Индексируемая таблица.
FTS$FIELD_NAME	VARCHAR(31) CHARACTER SET UNICODE_FSS	Индексируемое поле.

FTS$REINDEX

Служебная процедура FTS$REINDEX производит полную (по всем записям, с заменой ранее существующего индекса) переиндексацию указанного индекса.

Таблица 25.12 — Входные параметры процедуры FTS$REINDEX

Имя	Тип	Описание
FTS$INDEX_NAME	VARCHAR(31) CHARACTER SET UNICODE_FSS	Имя индекса.

FTS$SEARCH

Для извлечения данных из индекса используется процедура FTS$SEARCH. Обязательными входными параметрами являются имя индекса, по которому будет осуществляться поиск, и условие поиска.

Таблица 25.13 — Входные параметры процедуры FTS$SEARCH

Имя поля	Тип	Описание
FTS$INDEX_NAME	VARCHAR(31) CHARACTER SET UNICODE_FSS	Имя индекса.
FTS$RELATION_NAME	VARCHAR(31) CHARACTER SET UNICODE_FSS	Имя индексируемой таблицы. Может принимать значение NULL, тогда поиск будет идти по всем таблицам, входящим в индекс.
FTS$FILTER	VARCHAR(4000)	Фильтр, по которому будет осуществляться поиск (см. lucene.apache.org).
FRAGMENT_SIZE	INTEGER	Параметр, задающий отображаемое количество символов фрагмента текста, в котором находится строка, соответствующая условиям поиска. По умолчанию используется значение, равное 50.

Таблица 25.14 — Выходные параметры процедуры FTS$SEARCH

Имя поля	Тип	Описание
FTS$ROW_ID	CHAR(8) CHARACTER SET OCTETS	Значение RDB$DB_KEY для таблицы, содержащей запись.
FTS$SCORE	DOUBLE PRECISION	Оценка соответствия возвращаемой записи условию поиска.
FTS$RELATION	VARCHAR(31) CHARACTER SET UNICODE_FSS	Таблица, в которой найдено поле, удовлетворяющее фильтру.
FTS$HIGHLIGHT	VARCHAR(512)	Фрагмент текста, содержащий строку,удовлетворяющей условиям поиска. Найденная строка заключается в теги <B> </B>.

FTS$SEARCH_ID

Возвращает значение первичного ключа, если таблица полнотекстового поиска имеет только один первичный ключ и он целочисленный. При попытке вызова для индекса, включающего таблицу с составным первичным ключом или отличным от целочисленного, будет возвращаться ошибка.

Входные параметры процедуры FTS$SEARCH_ID такие же, как у FTS$SEARCH.

Таблица 25.15 — Выходные параметры процедуры FTS$SEARCH_ID

Имя поля	Тип	Описание
FTS$ROW_ID	BIGINT	Значение первичного ключа.
FTS$SCORE	DOUBLE PRECISION	Оценка соответствия возвращаемой записи условию поиска.
FTS$RELATION	VARCHAR(31) CHARACTER SET UNICODE_FSS	Таблица, в которой найдено поле, удовлетворяющее фильтру.
FTS$HIGHLIGHT	VARCHAR(512)	Фрагмент текста, содержащий строку,удовлетворяющей условиям поиска. Найденная строка заключается в теги <B> </B>.

FTS$SEARCH_JSON

Возвращает BLOB с JSON значением - имя таблицы, имена полей первичных ключей и их значения.

Входные параметры процедуры FTS$SEARCH_JSON такие же, как у FTS$SEARCH.

Таблица 25.16 — Выходные параметры процедуры FTS$SEARCH_JSON

Имя поля	Тип	Описание
FTS$ROW_ID	BLOB	JSON значение, содержащее имя таблицы, имена полей первичных ключей и их значения.
FTS$SCORE	DOUBLE PRECISION	Оценка соответствия возвращаемой записи условию поиска.
FTS$RELATION	VARCHAR(31) CHARACTER SET UNICODE_FSS	Таблица, в которой найдено поле, удовлетворяющее фильтру.
FTS$HIGHLIGHT	VARCHAR(512)	Фрагмент текста, содержащий строку,удовлетворяющей условиям поиска. Найденная строка заключается в теги <B> </B>.

FTS$FULL_REINDEX

Для выполнения полной переиндексации для всех индексов, созданных в БД, выполняется процедурой `FTS$FULL_REINDEX`. Процедура не имеет входных и выходных параметров.

FTS$STARTDAEMON

Запускает демон[3] переиндексации для того, чтобы при изменении индексируемых наборов данных переиндексация выполнялась автоматически. «Демон» выполняет непрерывный (каждые 0,1 с) мониторинг таблицы `FTS$POOL` и переиндексацию измененных записей, после чего соответствующие записи удаляются из `FTS$POOL`. Процедура не имеет входных и выходных параметров.

FTS$STOPDAEMON

Останавливает демон переиндексации. Процедура не имеет входных и выходных параметров.

25.4 Пример использования полнотекстового поиска

Допустим в базе данных создана таблица `TEST_TABLE`:

```
create table TEST_TABLE(
id int primary key,
str varchar(1024));
```

Добавим в нее несколько записей:

```
insert into TEST_TABLE values(0, 'There were photographs of all her children proudly
displayed on the mantelpiece.');
insert into TEST_TABLE values(1, 'His trophies were proudly displayed in a backlit
cabinet.');
insert into TEST_TABLE values(2, 'Characters in mathematical mode are usually shown in
italics, but sometimes especial function names require different formatting, this is
accomplished by using operators.');
```

И попытаемся найти в ней слово `'displayed'` полнотекстовым поиском.

В общем случае протокол использования системы полнотекстового поиска можно представить следующим образом:

1. создание индекса;
2. добавление/удаление полей в индекс;
3. выполнение переиндексации;
4. поиск;
5. удаление индекса (если необходимо).

25.4.1 Создание индекса

Сначала необходимо создать индекс. Для этого следует выполнить процедуру `FTS$CREATE_INDEX`, указав необходимые входные параметры. Обязательным входным параметром является только имя индекса. Второй входной параметр позволяет задать тип анализатора. Доступные типы анализаторов приведены в таблице 25.8. По умолчанию используется значение `Standard`. Значение третьего параметра задает описание для индекса.

[3] Демон — это процесс, работающий в фоновом режиме без прямого общения с пользователем.

```
EXECUTE PROCEDURE FTS$CREATE_INDEX('TEST_INDEX');
```

При этом в таблицу FTS$INDICES добавится запись. Значение поля FTS$INDEX_STATUS равно N, что означает, что индекс только что создан, требует полной переиндексации.

25.4.2 Добавление полей в индекс

После того как индекс создан, в него можно добавлять поля из таблиц базы данных. Для добавления поля в индекс используется процедура FTS$ADD_FIELD_TO_INDEX.

Процедура имеет следующие обязательные входные параметры: имя индекса, имя таблицы, имя поля.

```
EXECUTE PROCEDURE FTS$ADD_FIELD_TO_INDEX('TEST_INDEX', 'TEST_TABLE', 'STR');
```

После добавления поля в индекс в таблице FTS$INDEX_SEGMENTS должна появиться соответствующая запись.

25.4.3 Удаление полей из индекса

Для удаления полей из индекса в системе полнотекстового писка используется процедура FTS$DROP_FIELD_FROM_INDEX. Процедура имеет три обязательных входных параметра: имя индекса, из состава которого удаляется поле; имя таблицы, которая содержит это поле; имя удаляемого поля.

```
EXECUTE PROCEDURE FTS$DROP_FIELD_FROM_INDEX('TEST_INDEX','TEST_TABLE','STR');
```

После удаления поля из состава из таблицы FTS$INDEX_SEGMENTS удалится запись, связывающая поле и индекс.

25.4.4 Переиндексация

Для того чтобы обновить данные индекса следует выполнить переиндексацию. Переиндексация может быть выполнена вызовом одной из трех процедур:

- FTS$FULL_REINDEX
- FTS$REINDEX
- FTS$STARTDAEMON

Процедура FTS$FULL_REINDEX выполняет полную переиндексацию для всех индексов в системе полнотекстового поиска. Процедура FTS$FULL_REINDEX не имеет входных параметров.

```
EXECUTE PROCEDURE FTS$FULL_REINDEX;
```

Процедура FTS$REINDEX позволяет выполнить полную переиндексацию по указанному индексу. Процедура имеет один обязательный входной параметр — имя индекса, например:

```
EXECUTE PROCEDURE FTS$REINDEX ('TEST_INDEX');
```

Процедура FTS$STARTDAEMON запускает «демон» переиндексации. «Демон» выполняет каждые 0.1 секунды мониторинг таблицы FTS$POOL. В таблице FTS$POOL сохраняются RDB$DB_KEY изменившихся записей. По значению RDB$DB_KEY «демон» выполняет переиндексацию только изменившихся записей, после чего RDB$DB_KEY переиндексированных записей удаляется из таблицы FTS$POOL. «Демон» следует запускать в отдельном коннекте к базе.

```
EXECUTE PROCEDURE FTS$STARTDAEMON;
```

25.4.5 Поиск

Для извлечения данных из индекса в системе полнотекстового поиска используется процедура FTS$SEARCH.

Процедура имеет следующие входные параметры: имя индекса, имя таблицы, фраза поиска и отображаемое количество символов результата поиска. Второй входной параметр – имя индексируемой таблицы может принимать значение NULL, в этом случае поиск выполняется по всем таблицам, входящим в индекс. Если же указано имя индексируемой таблицы, то поиск будет выполняться только по этой таблице.

Выходные параметры процедуры: таблица, в которой найдены данные, удовлетворяющие условию поиска; значение RDB$DB_KEY для найденных записей; значение соответствия найденной записи условию поиска; фрагмент текста с искомой строкой.

Пример поиска слова 'displayed' по всем таблицам в индексе TEST_INDEX, длина фрагмента результата поиска равна 20 символов:

```
SELECT * from FTS$SEARCH('TEST_INDEX', NULL, 'displayed', 20);
```

Допустим, что при поиске было найдено две записи, в этом случае результат поиска для приведенного выше примера может иметь вид:

ROW_ID	SCORE	RELATION	HIGHLIGHT
840000000A000000	0.5414568781852722	TEST_TABLE	His trophies were proudly <B>displayed</B>
8400000009000000	0.5178392529487610	TEST_TABLE	<B>displayed</B> on the mantelpiece.

Здесь ROW_ID это значение RDB$DB_KEY найденной записи; SCORE – соответствие найденной записи условию поиска; RELATION – имя таблицы, в которой была найдена запись; HIGHLIGHT – фрагмент текста, содержащий строку, удовлетворяющую условиям поиска.

По значению RDB$DB_KEY можно выбрать непосредственно найденные записи из соответствующих таблиц, например:

```
SELECT B.* from FTS$SEARCH('TEST_INDEX', NULL, 'displayed', 20) as A
   LEFT JOIN TEST_TABLE as B
      ON A.FTS$ROW_ID = B.RDB$DB_KEY;
```

Результатом такого запроса в отличие от предыдущего примера будет выборка из таблицы TEST_TABLE, например:

ID	STR
1	His trophies were proudly displayed in a backlit cabinet.
0	There were photographs of all her children proudly displayed on the mantelpiece.

Пример, возвращающий в FTS$ROW_ID JSON значение:

```
SELECT FTS$ROW_ID, FTS$SCORE, FTS$RELATION, FTS$HIGHLIGHT
FROM FTS$SEARCH_JSON('TEST_INDEX', NULL, 'displayed', 100);
```

Результат запроса:

ROW_ID	SCORE	RELATION	HIGHLIGHT
{"data":[{"ID":1}], "table":"TEST_TABLE"}	0.253744274377822	TEST_TABLE	His trophies were proudly <B>displayed</B> in a backlit cabinet.
{"data":[{"ID":0}], "table":"TEST_TABLE"}	0.230805337429046	TEST_TABLE	There were photographs of all her children proudly <B>displayed</B> on the mantelpiece.

Пример, возвращающий в FTS$ROW_ID ID записи таблицы:

```
SELECT FTS$ROW_ID, FTS$SCORE, FTS$RELATION, FTS$HIGHLIGHT
FROM FTS$SEARCH_ID('TEST_INDEX', NULL, 'displayed', 100);
```

Результат запроса:

ROW_ID	SCORE	RELATION	HIGHLIGHT
1	0.2537442743778229	TEST_TABLE	His trophies were proudly <B>displayed</B> in a backlit cabinet.
0	0.2308053374290466	TEST_TABLE	There were photographs of all her children proudly <B>displayed</B> on the mantelpiece.

Ошибка, возвращаемая процедурой FTS$SEARCH_ID, когда индекс содержит более одного первичного ключа:

```
Statement failed, SQLSTATE = HY000
java.sql.SQLException: Table 'TEST_TABLE' has more than 1 field in the primary key
at org.firebirdsql.lucene.query.LuceneResultSet.fetch(LuceneResultSet.java:157)
-At procedure 'FTS$SEARCH_ID'
```

25.4.6 Индексация изменений в таблице

Для поддержания актуальности полнотекстовых индексов таблиц, задействованных в полнотекстовом поиске, необходимо указать плагин FTS в plugins.conf:

```
Plugin = FTS {
    Module = $(dir_plugins)/fts
    Config = FTS_config
}

Config = FTS_config {
    JarDirs = $(this)/jar/fts
}
```

И настроить плагин репликации FTS в конфигурационном файле replication.conf:

```
database = /examples/empbuild/employee.fdb
{
    plugin = FTS
}
```

Если на сервере настроена репликация, то для включения полнотекстовой репликации также необходимо указать FTS плагин. К примеру, для мастер-базы настройка будет выглядеть следующим образом:

```
database = /db/employee.fdb
{
    sync_replica = sysdba:masterkey@localhost:/db/fts_repl.fdb
    plugin = FTS
}
```

Для базы-реплики также необходимо указать плагин FTS, кроме того, должна быть включена настройка каскадной репликации. Например:

```
database = /db/fts_repl.fdb
{
    cascade_replication = true
    plugin = FTS
}
```

Для полнотекстовой индексации таблиц необходимо включить репликацию в базе:

```
ALTER DATABASE ENABLE PUBLICATION;
```

А также добавить задействованные в полнотекстовом поиске таблицы в публикации:

```
ALTER DATABASE INCLUDE TABLE TABLE1, TABLE2,... TO PUBLICATION;
```

Без этого плагин будет неактивен.

25.4.7 Удаление индекса

Для удаления индекса используется процедура FTS$DROP_INDEX, которая имеет один обязательный входной параметр – имя индекса.

```
EXECUTE PROCEDURE FTS$DROP_INDEX ('TEST_INDEX');
```

При удалении индекса удаляются:

- Соответствующая запись из таблицы FTS$INDICES;
- Связанные с индексом записи из таблицы FTS$INDEX_SEGMENTS;
- Файлы индекса на диске.

25.5 Синтаксис поисковых запросов

25.5.1 Термы

Поисковые запросы (фразы поиска) состоят из термов и операторов. Lucene поддерживает простые и сложные термы. Простые термы состоят из одного слова, сложные из нескольких. Первые из них, это обычные слова, например, "привет", "тест". Второй же тип термов это группа слов, например, "Привет как дела". Несколько термов можно связывать вместе при помощи логических операторов. Регистр символов неважен.

25.5.2 Маска

Lucene позволяет производить поиск документов по маске, используя в термах символы «?» и «*». В этом случае символ «?» заменяет один любой символ, а «*» - любое количество символов, например:

```
'te?t'
'test*'
'tes*t'
```

Поисковый запрос нельзя начинать с символов «?» или «*».

25.5.3 Нечеткий поиск

Для выполнения нечеткого поиска в конец терма следует добавить тильду «~». В этом случае будут искаться все похожие слова, например при поиске "roam~" будут так же найдены слова "foam" и "roams".

25.5.4 Усиление термов

Lucene позволяет изменять значимость термов во фразе поиска. Например, вы ищете фразу "Hello world" и хотите, чтобы слово «world» было более значимым. Значимость терма во фразе поиска можно увеличить, используя символ «^», после которого указывается коэффициент усиления. В следующем примере значимость слова «world» в четыре раза больше значимости слова «Hello», которая по умолчанию равна единице.

```
'Hello world^4'
```

25.5.5 Логические операции

Булевы операторы позволяют использовать логические конструкции при задании условий поиска, позволяют комбинировать несколько термов. Lucene поддерживает следующие булевы операторы: AND, +, OR, NOT, -.

Булевы операторы должны указываться заглавными буквами.

Оператор OR

OR является булевым оператором по умолчанию, это означает, что если между двумя термами фразы поиска не указан другой булев оператор, то подставляется оператор OR. При этом система поиска находит документ, если одна из указанных во фразе поиска терм в нем присутствует. Альтернативным обозначением оператора OR является «||».

```
'Hello world world'
```

Эквивалентно:

```
'Hello world OR world'
```

Оператор AND

Оператор `AND` указывает на то, что в тексте должны присутствовать все, объединенные оператором термы поиска. Альтернативным обозначением оператора является «`&&`».

```
'Hello AND world'
```

Оператор +

Оператор `+` указывает на то, что следующее за ним слово должно обязательно присутствовать в тексте (после знака `+` не должно быть пробела). Например, для поиска записей, которые должны содержать слово «`hello`» и могут содержать слово «`world`», фраза поиска может иметь вид:

```
'+Hello world'
```

Оператор NOT

Оператор `NOT` позволяет исключить из результатов поиска те, в которых встречается терм, следующий за оператором. Вместо слова `NOT` может использоваться символ «`!`». Например, для поиска записей, которые должны содержать слово «`hello`» и не должны содержать слово «`world`», фраза поиска может иметь вид:

```
'Hello NOT world'
```

Оператор `NOT` не может использоваться только с одним термом. Например, поиск с таким условием не вернет результатов:

```
'NOT world'
```

Оператор —

Этот оператор является аналогичным оператору `NOT`. После знака `-` не должно быть пробела. Пример использования:

```
'Hello -world'
```

Группировка булевых операторов

Анализатор запросов `Lucene` поддерживает группировку булевых операторов. Допустим, требуется найти либо слово «`word`», либо слово «`dolly`» и обязательно слово «`hello`», для этого используется такой запрос:

```
'Hello && (world || dolly)'
```

25.5.6 Экранирование специальных символов

Для включения специальных символов во фразу поиска выполняется их экранирование обратным слешем «\». Ниже приведен список специальных символов, используемых в Lucene на данный момент:

```
+ - && || ! ( ) { } [ ] ^ " ~ * ? : \
```

Фраза поиска для выражения $(1 + 1) : 2$ будет иметь вид:

```
'\(1\+1\)\:2'
```

Более подробное англоязычное описание синтаксиса расположено на официальном сайте Lucene.

Глава 26

Операторы обеспечения безопасности

26.1 Операторы управления пользователями

В данном разделе описываются операторы создания, модификации и удаления учётных записей пользователей средствами операторов SQL. Такая возможность предоставлена следующим пользователям:

- SYSDBA;
- Любому пользователю, имеющему права на роль RDB$ADMIN в базе данных пользователей и права на ту же роль для базы данных в активном подключении. Пользователь должен подключаться к базе данных с ролью RDB$ADMIN или получить её права, если роль назначена в качестве роли по умолчанию;
- Любому пользователю с ролью, которой назначена системная привилегия USER_MANAGEMENT в базе данных безопасности (о системных привилегиях см. раздел 26.2). Пользователь должен подключаться к базе данных с этой ролью или получить её права, если роль назначена в качестве роли по умолчанию;
- При включенном флаге AUTO ADMIN MAPPING в базе данных пользователей (security5.fdb или той, что установлена для вашей базы данных в файле databases.conf) — любой администратор операционной системы Windows (при условии использования сервером доверенной авторизации — trusted authentication) без указания роли. При этом не важно, включен или выключен флаг AUTO ADMIN MAPPING в самой базе данных.

Непривилегированные пользователи могут использовать только оператор ALTER USER для изменения собственной учётной записи.

26.1.1 CREATE USER

Оператор CREATE USER создаёт учётную запись пользователя РЕД Базы Данных. Пользователь должен отсутствовать в текущей базе данных безопасности иначе будет выдано соответствующее сообщение об ошибке.

Листинг 26.1. Синтаксис оператора CREATE USER

```
CREATE USER <логин> PASSWORD '<пароль>'
   [FIRSTNAME '<имя пользователя>']
   [MIDDLENAME '<отчество пользователя>']
   [LASTNAME '<фамилия пользователя>']
   [ACTIVE | INACTIVE]
   [USING PLUGIN 'имя плагина']
   [TAGS (<атрибут> [, <атрибут> ...] )]
   [GRANT ADMIN ROLE]

<атрибут> ::= <имя атрибута> = 'строковое значение'
```

Начиная с версии 3.0 имена пользователей подчиняются общему правилу наименования идентификаторов объектов метаданных. Таким образом, пользователь с именем "Alex" и с именем "ALEX" будут разными пользователями.

Предложение PASSWORD задаёт пароль пользователя. Максимальная длина пароля зависит от того какой менеджер пользователей задействован (параметр UserManager в файле конфигурации firebird.conf). Для менеджер пользователей SRP эффективная длина пароля ограничена 20 байтами. Для ме-

неджер пользователей Legacy_UserManager максимальная длина пароля равна 8 байт.

Необязательные предложения FIRSTNAME, MIDDLENAME и LASTNAME задают дополнительные атрибуты пользователя, такие как имя пользователя, отчество и фамилия соответственно. Максимальная длина 32 символа.

Кроме того можно задать неограниченное количество пользовательских атрибутов с помощью необязательного предложения TAGS. Данная возможность доступна только при использовании Srp в качестве менеджера пользователей.

Если при создании учётной записи будет указан атрибут INACTIVE, то пользователь будет создан в "неактивном состоянии", т.е. подключиться с его учётной записью будет невозможно. При указании атрибута ACTIVE пользователь будет создан в активном состоянии (по умолчанию). Данная возможность доступна только при использовании Srp в качестве менеджера пользователей.

С опцией GRANT ADMIN ROLE создаётся новый пользователь с правами роли RDB$ADMIN в базе данных пользователя (security5.fdb). Это позволяет ему управлять учётными записями пользователей, но не дает ему специальных полномочий в обычных базах данных.

Необязательное предложение USING PLUGIN позволяет явно указывать какой плагин управления пользователями будет использован. По умолчанию используется тот плагин, который был указан первым в списке параметра UserManager в файле конфигурации firebird.conf. Допустимыми являются только значения, перечисленные в параметре UserManager. Следует учитывать, что одноименные пользователи, созданные с помощью разных плагинов управления пользователями — это разные пользователи.

Если предложение USING PLUGIN не указано, то при добавлении пользователя он сам добавляется во все плагины из списка параметра DefaultUserManagers (в том числе его атрибуты).

```
CREATE USER jon PASSWORD '87654321'
FIRSTNAME 'Jon'
LASTNAME 'Snow'
TAGS (BIRTHYEAR = '283' , CITY = 'Winterfell');
```

Подробнее о работе с пользователями см. в «Руководстве администратора».

26.1.2 ALTER USER

Для изменения существующей учетной записи пользователя используется следующий синтаксис:

Листинг 26.2. Синтаксис оператора ALTER USER

```
ALTER {USER <логин> | CURRENT USER}
{
   [SET]
   [PASSWORD '<пароль>']
   [FIRSTNAME '<имя пользователя>']
   [MIDDLENAME '<отчество пользователя>']
   [LASTNAME '<фамилия пользователя>']
   [ACTIVE | INACTIVE]
   [TAGS (<атрибут>|DROP <имя атрибута> [, <атрибут>|DROP <имя атрибута>...] )]
}
[USING PLUGIN 'имя плагина']
[{GRANT | REVOKE} ADMIN ROLE];

<атрибут> ::= <имя атрибута> = 'строковое значение'
```

В операторе ALTER USER должно присутствовать хотя бы одно из необязательных предложений.

Это единственный оператор управления учётными записями, который может также использоваться непривилегированными пользователями для изменения их собственных учетных записей, однако это не относится к опциям GRANT/REVOKE ADMIN ROLE и атрибуту ACTIVE/INACTIVE для изменения которых, необходимы административные привилегии. Если требуется изменить свою учётную запись, то вместо указания имени текущего пользователя можно использовать предложение CURRENT USER.

Необязательное предложение PASSWORD задаёт новый пароль пользователя.

Необязательные предложения FIRSTNAME, MIDDLENAME и LASTNAME позволяют изменить дополнительные атрибуты пользователя, такие как имя пользователя (имя человека), отчество и фамилия соответственно.

Атрибут INACTIVE позволяет сделать учётную запись неактивной. Это удобно когда необходимо временно отключить учётную запись без её удаления. Атрибут ACTIVE позволяет вернуть неактивную учётную запись в активное состояние. Данная возможность доступна только при использовании Srp в качестве менеджера пользователей.

Необязательное предложение TAGS позволяет задать, изменить или удалить пользовательские атрибуты. Если в списке атрибутов, атрибута с заданным именем не было, то он будет добавлен, иначе его значение будет изменено. Атрибуты не указанные в списке не будут изменены. Для удаления пользовательского атрибута перед его именем в списке атрибутов необходимо указать ключевое слово DROP. Данная возможность доступна только при использовании Srp в качестве менеджера пользователей.

Предложение GRANT ADMIN ROLE предоставляет указанному пользователю привилегии роли RDB$ADMIN в текущей базе данных безопасности. Это позволяет указанному пользователю управлять учётными записями пользователей, но не даёт ему специальных полномочий в обычных базах данных.

Предложение REVOKE ADMIN ROLE отбирает у указанного пользователя привилегии роли RDB$ADMIN в текущей базе данных безопасности. Это запрещает указанному пользователю управлять учётными записями пользователей.

Необязательное предложение USING PLUGIN позволяет явно указывать какой плагин управления пользователями будет использован. По умолчанию используется тот плагин, который был указан первым в списке параметра UserManager в файле конфигурации firebird.conf. Допустимыми являются только значения, перечисленные в параметре UserManager.

Если предложение USING PLUGIN не указано, то при изменении атрибутов пользователя они сами меняется у соответствующих пользователейво во всех плагинах из списка параметра DefaultUserManagers.

Если в каком-либо стандартном плагине нет пользователя, то он добавляется, но только есть среди изменяемых атрибутов есть пароль.

26.1.3 CREATE OR ALTER USER

Оператор CREATE OR ALTER USER создаёт новую или изменяет учётную запись.

Листинг 26.3. Синтаксис оператора CREATE OR ALTER USER

```
CREATE OR ALTER USER <логин>
{
   [SET]
   [PASSWORD '<пароль>']
   [FIRSTNAME '<имя пользователя>']
   [MIDDLENAME '<отчество пользователя>']
   [LASTNAME '<фамилия пользователя>']
   [ACTIVE | INACTIVE]
   [TAGS (<атрибут>|DROP <имя атрибута> [, <атрибут>|DROP <имя атрибута>...] )]
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
}
[USING PLUGIN 'имя плагина']
[{GRANT | REVOKE} ADMIN ROLE];

<атрибут> ::= <имя атрибута> = 'строковое значение'
```

Если пользователя не существует, то он будет создан с использованием предложения `CREATE USER`. Если он уже существует, то он будет изменён, при этом существующие привилегии сохраняются.

> Если предложение `USING PLUGIN` не указано, то при создании пользователя он сам добавляется во все плагины из списка параметра `DefaultUserManagers` (в том числе его атрибуты).
>
> При изменении пароля пользователя он сам меняется во всех плагинах из списка параметра `DefaultUserManagers`. Если в каком-то плагине нет пользователя, то он добавляется.
>
> Если меняется какой-либо другой атрибут, то он также меняется и в других плагинах, но если пользователь отсутствует, то он не создаётся.

Семантика операторов и предложений в этом операторе полностью соответствует оператору `ALTER USER`

26.1.4 DROP USER

Для удаления существующей учетной записи пользователя используется следующий синтаксис:

Листинг 26.4. Синтаксис оператора `DROP USER`

```
DROP USER <логин>
[USING PLUGIN 'имя плагина'];
```

Необязательное предложение `USING PLUGIN` позволяет явно указывать какой плагин управления пользователями будет использован. По умолчанию используется тот плагин, который был указан первым в списке параметра `UserManager` в файле конфигурации `firebird.conf`. Допустимыми являются только значения, перечисленные в параметре `UserManager`. Следует учитывать, что одноименные пользователи, созданные с помощью разных плагинов управления пользователями — это разные пользователи.

26.1.5 RECREATE USER

Оператор `RECREATE USER` создаёт нового или пересоздаёт существующего пользователя.

Листинг 26.5. Синтаксис оператора `RECREATE USER`

```
RECREATE USER <логин> PASSWORD '<пароль>'
   [FIRSTNAME '<имя пользователя>']
   [MIDDLENAME '<отчество пользователя>']
   [LASTNAME '<фамилия пользователя>']
   [ACTIVE | INACTIVE]
   [TAGS (<атрибут>|DROP <имя атрибута> [, <атрибут>|DROP <имя атрибута>...] )]
   [USING PLUGIN 'имя плагина']
   [GRANT ADMIN ROLE];

<атрибут> ::= <имя атрибута> = 'строковое значение'
```

Если пользователь с таким именем уже существует, то оператор RECREATE USER удалит его и создаст нового. Существующие привилегии при этом будут сохранены.

Если предложение USING PLUGIN не указано, то при добавлении пользователя он сам добавляется во все плагины из списка параметра DefaultUserManagers (в том числе атрибуты).

Семантика операторов и предложений в этом операторе полностью соответствует оператору CREATE USER.

26.2 Операторы управления ролями

Роль (role) — объект базы данных, представляющий набор привилегий. Множество привилегий предоставляется роли, а затем роль может быть предоставлена или отозвана у одного или нескольких пользователей.

Пользователь, которому предоставлена роль, должен указать её при входе, для того чтобы получить её привилегии, или же эта роль должна быть грантована с использованием ключевого слова DEFAULT. Любые другие привилегии, предоставленные пользователю, не будут затронуты при его входе в систему с указанной ролью. Вход в систему с несколькими ролями не поддерживается, однако вы можете права нескольких ролей назначенных по умолчанию. Вы можете изменить текущую роль с помощью оператора SET ROLE.

Роли могут быть грантованы другие роли. При входе с этой ролью пользователь автоматически получит права всех ролей выданных с использованием ключевого слова DEFAULT.

26.2.1 CREATE ROLE

Оператор позволяет создать новую роль. Синтаксис оператора:

Листинг 26.6. Синтаксис оператора CREATE ROLE

```
CREATE ROLE <имя роли>
[SET SYSTEM PRIVILEGES TO <сис.привилегия> [,<сис.привилегия> ...]];
```

Создать новую роль может администратор и пользователь с привилегией CREATE ROLE.

Имя роли может содержать до 63 символов и должно быть уникальным среди всех имен ролей.

Желательно также чтобы имя роли было уникальным не только среди имён ролей, но и среди имён пользователей. Если вы создадите роль с тем же именем существующего пользователя, то такой пользователь не сможет подключиться к базе данных.

Ролям могут предоставляться привилегии к объектам базы данных точно так же, как и пользователям. Для этого используется оператор GRANT. Одной роли может быть предоставлено произвольное количество привилегий. В дальнейшем роли могут назначаться отдельным пользователям, которые в результате получают все привилегии, предоставленные роли. Одна роль может быть назначена любому количеству пользователей. Роль, под которой пользователь соединяется с базой данных, задается в операторе CONNECT.

Предложение SET SYSTEM PRIVILEGES TO позволяет создать роль с системными привилегиями. Системные привилегии это части привилегий администратора. Таким образом, через делегирование роли с системными привилегиями пользователю можно передавать ему часть прав администратора БД.

Системные привилегии позволяют производить очень тонкую настройку, поэтому иногда вам нужно будет выдать более 1 системной привилегии для выполнения какой-либо задачи. Например, необходимо выдать IGNORE_DB_TRIGGERS совместно с USE_GSTAT_UTILITY, потому что gstat должен игнорировать триггера на события БД.

Таблица 26.1 — Системные привилегии

Привилегия	Описание
USER_MANAGEMENT	Управление пользователями.
READ_RAW_PAGES	Чтение страниц в сыром формате используя Attachment::getInfo().
CREATE_USER_TYPES	Создание, изменение и удаление не системных записей в таблице RDB$USER_TYPES.
USE_NBACKUP_UTILITY	Использование nbackup для создания резервных копий.
CHANGE_SHUTDOWN_MODE	Закрытие базы данных (shutdown) и возвращение её в online.
TRACE_ANY_ATTACHMENT	Трассировка чужих пользовательских сессий.
MONITOR_ANY_ATTACHMENT	Мониторинг (MON$ таблицы) чужих пользовательских сессий.
CREATE_DATABASE	Создание новой базы данных (хранится в базе данных пользователей security.db).
DROP_DATABASE	Удаление текущей БД.
USE_GBAK_UTILITY	Использование утилиты или сервиса gbak.
USE_GSTAT_UTILITY	Использование утилиты или сервиса gstat.
USE_GFIX_UTILITY	Использование утилиты или сервиса gfix.
IGNORE_DB_TRIGGERS	Разрешает игнорировать триггеры на события БД.
CHANGE_HEADER_SETTINGS	Изменение параметров на заголовочной странице БД.
SELECT_ANY_OBJECT_IN_DATABASE	Выполнение оператора SELECT из всех селективных объектов (таблиц, представлений, хранимых процедур выбора).
ACCESS_ANY_OBJECT_IN_DATABASE	Доступ (любым способом) к любому объекту БД.
MODIFY_ANY_OBJECT_IN_DATABASE	Изменение любого объекта БД.
CHANGE_MAPPING_RULES	Изменение правил отображения при аутентификации.
USE_GRANTED_BY_CLAUSE	Использование GRANTED BY в операторах GRANT и REVOKE.
GRANT_REVOKE_ON_ANY_OBJECT	Выполнение операторов GRANT и REVOKE для любого объекта БД.
GRANT_REVOKE_ANY_DDL_RIGHT	Выполнение операторов GRANT и REVOKE для выдачи DDL привилегий.
CREATE_PRIVILEGED_ROLES	Создание привилегированных ролей (с использование SET SYSTEM PRIVILEGES).
GET_DBCRYPT_KEY_NAME	Получение имени ключа шифрования.
MODIFY_EXT_CONN_POOL	Управление пулом внешних соединений.
REPLICATE_INTO_DATABASE	Использование API репликации для загрузки наборов изменений в базу данных.

Для проверки имеет ли текущее подключение заданную системную привилегию можно воспользоваться встроенной функцией RDB$SYSTEM_PRIVILEGE.

```
CREATE ROLE SYS_UTILS
SET SYSTEM PRIVILEGES TO USE_GBAK_UTILITY, USE_GSTAT_UTILITY, IGNORE_DB_TRIGGERS;
```

26.2.2 ALTER ROLE

Оператор ALTER ROLE изменяет список системных привилегий роли или удаляет их.

Листинг 26.7. Синтаксис оператора ALTER ROLE

```
ALTER ROLE
{
      <имя роли> SET SYSTEM PRIVILEGES TO <сис.привилегия> [,<сис.привилегия>...]
    | <имя роли> DROP SYSTEM PRIVILEGES
    | RDB$ADMIN {SET | DROP} AUTO ADMIN MAPPING
}
```

При использовании предложения SET SYSTEM PRIVILEGES TO роли назначаются системные привилегии из списка. Для очистки списка системных привилегий установленных предыдущим оператором используйте оператор ALTER ROLE с предложением DROP SYSTEM PRIVILEGES.

Выполнить оператор могут администраторы, владелец роли или пользователи с привилегией ALTER ANY ROLE.

26.2.3 ALTER ROLE RDB$ADMIN

Оператор ALTER ROLE RDB$ADMIN включает или отключает возможности администраторам Windows автоматически получать привилегии роли RDB$ADMIN при входе.

Листинг 26.8. Синтаксис оператора ALTER ROLE RDB$ADMIN

```
ALTER ROLE RDB$ADMIN {SET | DROP} AUTO ADMIN MAPPING
```

Этот оператор может быть выполнен владельцем базы данных или администратором.

Администраторы операционной системы Windows автоматически не получают права SYSDBA при подключении к базе данных. Имеют ли администраторы автоматические права SYSDBA зависит от установки значения флага AUTO ADMIN MAPPING.

Оператор ALTER ROLE разрешает или запрещает автоматическое предоставление роли RDB$ADMIN администраторам Windows в текущей базе данных, если используется доверительная авторизация (Trusted Authentication). По умолчанию автоматическое предоставление роли RDB$ADMIN отключено.

В настоящее время AUTO ADMIN MAPPING является устаревшим и поддерживается для обратной совместимости, вместо него рекомендуется использовать операторы {CREATE | ALTER | DROP} MAPPING.

26.2.4 DROP ROLE

Оператор DROP ROLE удаляет существующую роль.

Листинг 26.9. Синтаксис оператора DROP ROLE

```
DROP ROLE <имя роли>
```

При удалении роли все привилегии, предоставленные этой роли, отменяются.

Выполнить оператор могут администраторы, владелец роли, пользователи с привилегией DROP ANY ROLE.

26.3 Операторы отображения объектов безопасности

26.3.1 CREATE MAPPING

Оператор CREATE MAPPING создаёт отображение объектов безопасности (пользователей, групп, ролей) одного или нескольких плагинов аутентификации на внутренние объекты безопасности – CURRENT_USER и CURRENT_ROLE.

Листинг 26.10. Синтаксис оператора CREATE MAPPING

```
CREATE [GLOBAL] MAPPING <имя отображения>
USING {
   PLUGIN <имя плагина> [IN <имя базы данных>]
 | ANY PLUGIN [IN <имя базы данных> | SERVERWIDE]
 | MAPPING [IN <имя базы данных>]
 | '*' [IN <имя базы данных>] }
FROM { ANY <тип отображаемого объекта> | <тип отображаемого объекта> <имя
отображаемого объекта> }
TO { USER | ROLE } [<имя объекта, на которое произведено отображение>]
```

Имя отображения должно быть уникальным среди имён отображений.

Если присутствует опция GLOBAL, то отображение будет применено не только для текущей базы данных, но и для всех баз данных находящимся в том же кластере, в том числе и базы данных безопасности.

Одноименные глобальные и локальные отображение — разные объекты.

Глобальное отображение работает, если в качестве базы данных безопасности используется база данных РЕД База Данных 3 или более высокой версии. Если вы планируете использовать другую базу данных, например, для целей использования собственного поставщика, то вам необходимо создать таблицу в ней и назвать её RDB$MAP с той же структурой, что и RDB$MAP в базе данных РЕД База Данных 3 и дать доступ на запись только для SYSDBA.

Предложение USING описывает источник отображения. Оно имеет весьма сложный набор опций:

- явное указание имени плагина (опция PLUGIN) означает, что оно будет работать только с этим плагином;
- оно может использовать любой доступный плагин (опция ANY PLUGIN), даже если источник является продуктом предыдущего отображения;
- оно может быть сделано так, чтобы работать только с обще серверными плагинами (опция SERVERWIDE);
- оно может быть сделано так, чтобы работать только с результатами предыдущего отображения (опция MAPPING);
- вы можете опустить использование любого из методов, используя звёздочку (*) в качестве аргумента;
- оно может содержать имя базы данных (опция IN), из которой происходит отображение объекта FROM.

Предложение FROM описывает отображаемый объект. Оно принимает обязательный аргумент — тип объекта. Особенности:

- при отображении имён из плагинов, тип определяется плагином;
- при отображении продукта предыдущего отображения, типом может быть только USER и ROLE;
- если имя объекта будет указано явно, то оно будет учитываться при отображении;
- при использовании ключевого слова ANY будут отображены объекты с любыми именами данного типа.

В предложении TO указывается пользователь или роль, на которого будет произведено отображение. NAME является не обязательным аргументом. Если он не указан, то в качестве имени объекта будет использовано оригинальное имя из отображаемого объекта.

Воспользоваться оператором создания отображений может SYSDBA, владелец базы данных (если отображение локальное), пользователь с ролью RDB$ADMIN, пользователь root (Linux).

Пример 1.

Включение доступа определённому пользователю из другой базы данных к текущей базе данных под другим именем.

```
CREATE MAPPING FROM_RT
USING PLUGIN SRP IN "rt"
FROM USER U1 TO USER U2;
```

Пример 2.

Включение SYSDBA сервера (от основной базы данных безопасности) для доступа к текущей базе данных.

```
CREATE MAPPING DEF_SYSDBA
USING PLUGIN SRP IN "security.db"
FROM USER SYSDBA TO USER;
```

Пример 3.

Обеспечение гарантирование, что у пользователей, которые подключаются традиционным плагином аутентификации не слишком много прав.

```
CREATE MAPPING LEGACY_2_GUEST
USING PLUGIN legacy_auth
FROM ANY USER TO USER GUEST;
```

Пример 4.

Включение использования доверительной аутентификации Windows во всех базах данных, которые используют текущую базу данных безопасности.

```
CREATE GLOBAL MAPPING TRUSTED_AUTH
USING PLUGIN WIN_SSPI
FROM ANY USER TO USER;
```

Пример 5.

Включение SYSDBA подобного доступа для администраторов Windows в текущей базе данных.

```
CREATE MAPPING WIN_ADMINS
USING PLUGIN WIN_SSPI
FROM Predefined_Group
DOMAIN_ANY_RID_ADMINS
TO ROLE RDB$ADMIN;
```

Группа DOMAIN_ANY_RID_ADMINS не существует в Windows, но такое имя будет добавлено плагином win_sspi для обеспечения точной обратной совместимости.

26.3.2 ALTER MAPPING

Оператор ALTER MAPPING позволяет изменять любые опции существующего отображения.

Листинг 26.11. Синтаксис оператора ALTER MAPPING

```
ALTER [GLOBAL] MAPPING <имя отображения>
USING {
   PLUGIN <имя плагина> [IN <имя базы данных>]
 | ANY PLUGIN [IN <имя базы данных> | SERVERWIDE]
 | MAPPING [IN <имя базы данных>]
 | '*' [IN <имя базы данных>] }
FROM { ANY <тип отображаемого объекта> | <тип отображаемого объекта> <имя
отображаемого объекта> }
TO { USER | ROLE } [<имя объекта, на которое произведено отображение>]
```

Изменить отображение может SYSDBA, владелец базы данных (если отображение локальное), пользователь с ролью RDB$ADMIN или пользователь root (Linux).

26.3.3 CREATE OR ALTER MAPPING

Оператор CREATE OR ALTER MAPPING создаёт новое или изменяет существующее отображение. Если отображение не существует, то оно будет создано с использованием предложения CREATE MAPPING.

Листинг 26.12. Синтаксис оператора CREATE OR ALTER MAPPING

```
     CREATE OR ALTER [GLOBAL] MAPPING <имя отображения>
USING {
   PLUGIN <имя плагина> [IN <имя базы данных>]
 | ANY PLUGIN [IN <имя базы данных> | SERVERWIDE]
 | MAPPING [IN <имя базы данных>]
 | '*' [IN <имя базы данных>] }
FROM { ANY <тип отображаемого объекта> | <тип отображаемого объекта> <имя
отображаемого объекта> }
TO { USER | ROLE } [<имя объекта, на которое произведено отображение>]
```

Семантика операторов и предложений в этом операторе полностью соответствует оператору CREATE MAPPING.

26.3.4 DROP MAPPING

Оператор DROP MAPPING удаляет существующее отображение. Если указана опция GLOBAL, то будет удалено глобальное отображение.

Листинг 26.13. Синтаксис оператора DROP MAPPING

```
DROP [GLOBAL] MAPPING <имя отображения>
```

Удалить отображение может SYSDBA, владелец базы данных (если отображение локальное), пользователь с ролью RDB$ADMIN, пользователь root (Linux).

26.4 Операторы управления политиками

Политики безопасности (политики учетных записей) позволяют контролировать следующие параметры безопасности системы:

- сложность пароля при его задании;
- количество предыдущих паролей, которые не должен повторять вновь заданный;
- срок действия пароля;
- количество допустимых неудачных попыток аутентификации;
- количество одновременно открытых сессий пользователя;
- продолжительность простоя пользователя до отключения;
- период времени неиспользования учетных записей пользователей;
- требования к дополнительным факторам для прохождения аутентификации (цифровые сертификаты).

Политики безопасности работают со всеми известными методами аутентификации.

Для удобного просмотра всех созданных политик существует псевдотаблица безопасности SEC$POLICIES (см. Руководство администратора Приложение Б "Псевдотаблицы безопасности").

26.4.1 CREATE POLICY

Для создание политики используется оператор CREATE POLICY. Синтаксис этого оператора приведен ниже:

```
CREATE POLICY <имя политики> [AS <параметр>=<значение> [,<параметр>=<значение>...]]
```

Хотя сами политики хранятся в базе данных безопасности security5.fdb, создать их можно, соединившись с любой базой данных. Однако пользователь при этом должен иметь права на запись в базу данных безопасности security5.fdb.

Возможные параметры политик следующие:

- AUTH_FACTORS — факторы аутентификации:
  - SRP — пароль (верификатор) SRP
  - LEGACY — пароль Legacy
  - WIN_SSPI
  - GSS
  - CERTIFICATE — сертификат пользователя
  - GOSTPASSWORD — пароль по ГОСТ
- PSWD_NEED_CHAR — минимальное количество букв в пароле;

- PSWD_NEED_DIGIT — минимальное количество цифр в пароле;
- PSWD_NEED_DIFF_CASE — требование использования различных регистров букв в пароле;
- PSWD_MIN_LEN — минимальная длина пароля;
- PSWD_VALID_DAYS — срок действия пароля;
- PSWD_UNIQUE_COUNT — количество последних не повторяющихся паролей;
- MAX_FAILED_COUNT — количество неудачных попыток входа;
- MAX_SESSIONS — максимальное число одновременных подключений одного пользователя;
- MAX_IDLE_TIME — время бездействия, в секундах;
- MAX_UNUSED_DAYS — максимальное время неактивности учетных записей пользователя, в днях.

Следующий пример демонстрирует создание политики:

```
CREATE POLICY TestPolicy AS
AUTH_FACTORS = (SRP, LEGACY),
PSWD_NEED_CHAR = 5,
PSWD_NEED_DIGIT = 3,
PSWD_MIN_LEN = 8,
PSWD_NEED_DIFF_CASE = true,
PSWD_VALID_DAYS = 15,
PSWD_UNIQUE_COUNT = 5,
MAX_FAILED_COUNT = 5,
MAX_SESSIONS = 10,
MAX_IDLE_TIME = 1800,
MAX_UNUSED_DAYS = 45;
```

26.4.2 ALTER POLICY

Оператор ALTER POLICY используется для изменения политики безопасности.

```
ALTER POLICY <имя политики> AS <параметр>=<значение> [,<параметр>=<значение>...]
```

Чтобы изменить политику безопасности пользователь должен иметь права на запись в базу данных безопасности security5.fdb и успешно соединиться с любой базой данных.

Возможные параметры политик можно посмотреть в описании оператора CREATE POLICY.

26.4.3 DROP POLICY

Для удаления политики безопасности администратору необходимо соединиться с какой-либо базой данных. Для удаления политики используется оператор DROP POLICY. Синтаксис этого оператора приведен ниже:

```
DROP POLICY <имя политики>
```

26.5 Операторы управления привилегиями

После создания объекта, только пользователь создавший объект (его владелец) и администраторы имеют доступ к нему. Пользователю необходимы привилегии на каждый объект, к которому он должен получить доступ. Как правило, привилегии должны быть предоставлены явно пользователю владельцем объекта или администратором базы данных. Привилегии предоставляются оператором GRANT, а отзываются с помощью оператора REVOKE.

26.5.1 GRANT

Оператор предоставляет одну или несколько привилегий для объектов базы данных пользователям, ролям, хранимым процедурам, функциям, пакетам, триггерам и представлениям. Синтаксис оператора:

Листинг 26.14. Синтаксис оператора GRANT

```
GRANT <табличные привилегии> ON [TABLE] {<имя таблицы/представления>}
   TO <список получателей привилегий> [WITH GRANT OPTION]
   [{GRANTED BY|AS} [USER] <имя грантора>]

GRANT EXECUTE ON {PROCEDURE | FUNCTION | PACKAGE} <имя процедуры/функции/пакета>
   TO <список получателей привилегий> [WITH GRANT OPTION]
   [{GRANTED BY|AS} [USER] <имя грантора>]

GRANT USAGE ON {EXCEPTION <имя искл-я>|{GENERATOR | SEQUENCE} <имя генератора>}
   TO <список получателей привилегий> [WITH GRANT OPTION]
   [{GRANTED BY|AS} [USER] <имя грантора>]

GRANT {ALL [PRIVILEGES] | {CREATE|ALTER ANY|DROP ANY} [, {CREATE|ALTER ANY| DROP ANY}.
..] } <объект>
   TO <список получателей привилегий> [WITH GRANT OPTION]
   [{GRANTED BY|AS} [USER] <имя грантора>]

GRANT CREATE DATABASE TO {USER <имя пользователя>|ROLE <имя роли>|GROUP <имя группы в
Unix>} [,{USER <имя пользователя>|ROLE <имя роли>|GROUP <имя группы в Unix>}...]

GRANT {ALL [PRIVILEGES] | {ALTER|DROP} [,{ALTER|DROP}...]} DATABASE
   TO <список получателей привилегий> [WITH GRANT OPTION]
   [{GRANTED BY|AS} [USER] <имя грантора>]

GRANT [DEFAULT] <имя роли> [, [DEFAULT] <имя роли> ...]
   TO [USER]|[ROLE] <имя польз-я/роли> [, [USER]|[ROLE] <имя польз-я/роли>...]
   [WITH ADMIN OPTION] [{GRANTED BY | AS} [USER] <имя грантора>]

GRANT POLICY <имя_политики> TO <имя_пользователя>

<табличные привилегии> ::= ALL [PRIVILEGES] | <привилегия> [, <привилегия>...]

<привилегия> ::=   SELECT
                 | DELETE
                 | INSERT
                 | UPDATE [(<имя столбца [, <имя столбца>...]>)]
                 | REFERENCES [(<имя столбца [, <имя столбца>...]>)]

 <объект> ::=  TABLE
             | TABLESPACE
             | VIEW
             | PROCEDURE
             | FUNCTION
             | PACKAGE
             | GENERATOR
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
              | SEQUENCE
              | DOMAIN
              | EXCEPTION
              | ROLE
              | CHARACTER SET
              | COLLATION
              | FILTER
              | JOB

 <список получателей привилегий> ::= {<объект получатель>|<пользователь получатель>}
                                    [, {<объект получатель>|<пользователь получатель>}..
.]

 <объект получатель> ::=  PROCEDURE <имя процедуры>
                        | FUNCTION <имя функции>
                        | PACKAGE <имя пакета>
                        | TRIGGER <имя триггера>
                        | VIEW <имя представления>
                        | SYSTEM PRIVILEGE <системная привилегия> }

 <пользователь получатель> ::=  [USER] <имя пользователя>
                              | [ROLE] <имя роли>
                              | GROUP <имя группы в Unix>
```

Все привилегии по доступу к объектам базы данных хранятся в самой базе, и не могут быть применены к любой другой базе данных.

Авторизованный пользователь не имеет никаких привилегий до тех пор, пока какие либо права не будут предоставлены ему явно. SYSDBA или владелец объекта могут выдавать привилегии другим пользователям, в том числе и привилегии на право выдачи привилегий другим пользователям.

Оператор позволяет выполнить одно из следующих действий:

- предоставить одну или несколько привилегий для таблиц и представлений пользователям, ролям, представлениям, хранимым процедурам, триггерам, пакетам и функциям;
- выдать права на выполнение процедуры, функции или пакета пользователям, ролям, представлениям, хранимым процедурам, триггерам, пакетам и функциям;
- контролирует доступ к исключениям, последовательностям и генераторам;
- предоставить одну или несколько привилегий на выполнение DDL операций над основными объектами базы данных пользователям, ролям, представлениям, хранимым процедурам, триггерам, пакетам и функциям;
- назначает привилегии на создание, удаление и изменение базы данных пользователям, ролям, представлениям, хранимым процедурам, триггерам, пакетам и функциям;
- назначить указанные роли группе перечисленных пользователей;
- назначить политику конкретному пользователю.

Предложение TO

В предложении TO указывается список пользователей, ролей и объектов базы данных (процедур, функций, пакетов, триггеров и представлений) для которых будут выданы перечисленные привилегии. Необязательные предложения USER и ROLE позволяют уточнить, кому именно выдаётся привилегия. Если ключевое слово USER или ROLE не указано, то сервер проверяет, существует ли роль с данным

именем, если таковой не существует, то привилегии назначаются пользователю. Существование пользователя, которому выдаются права, не проверяются при выполнении оператора GRANT. Если привилегия выдаётся объекту базы данных, то необходимо обязательно указывать тип объекта.

WITH GRANT OPTION

При предоставлении привилегий пользователям можно указать предложение WITH GRANT OPTION, что позволяет в свою очередь предоставлять другим пользователям эти привилегии.

Предложение GRANTED BY

С помощью предложение GRANTED BY можно предоставлять права не от имени текущего пользователя, а от другого пользователя. При использовании оператора REVOKE после GRANTED BY права будут удалены только в том случае, если они были зарегистрированы от удаляющего пользователя. Предложение AS является синонимом GRANTED BY. Предложения GRANTED BY и AS могут использовать владелец базы данных; SYSDBA; любой пользователь, имеющий права на роль RDB$ADMIN и указавший её при соединении с базой данных; при использовании флага AUTO ADMIN MAPPING — любой администратор операционной системы Windows (при условии использования сервером доверенной авторизации — trusted authentication), даже без указания роли. Даже владелец объекта не может использовать их, если он не имеет административных привилегий.

Для различных типов объектов метаданных существует различный набор привилегий. Эти привилегии будут описаны далее отдельно для каждого из типов объектов метаданных.

Привилегии для таблиц и представлений

Для таблиц и представлений возможно использовании сразу нескольких привилегий.

- SELECT — разрешает выборку данных из таблицы или представления. Можно указать ограничения, чтобы можно было изменять только указанные столбцы;
- DELETE — разрешает удалять записи из таблицы или представления;
- INSERT — разрешает добавлять записи в таблицу или представление;
- UPDATE — разрешает изменять записи в таблице или представлении. Можно указать ограничения, чтобы можно было изменять только указанные столбцы;
- REFERENCES — разрешает ссылаться на указанные столбцы внешним ключом. Необходимо указать для столбцов, на которых построен первичный ключ таблицы, если на неё есть ссылка внешним ключом другой таблиц;
- ALL [PRIVILEGES] — объединяет привилегии SELECT, INSERT, UPDATE, DELETE и REFERENCES.

Привилегии EXECUTE

Привилегия EXECUTE применима к хранимым процедурам, хранимым функциям, пакетам и унаследованным внешним функциям (UDF).

Для хранимых процедур привилегия EXECUTE позволяет не только выполнять хранимые процедуры, но и делать выборку данных из процедур выбора (с помощью оператора SELECT).

Привилегии USAGE

Для использования объектов метаданных, отличных от таблиц, представлений, хранимых процедур и функций, триггеров и пакетов, в пользовательских запросах необходимо предоставить пользователю привилегию USAGE для этих объектов.

Привилегия USAGE проверяется только для исключений и генераторов/последовательностей (в GEN_ID или NEXT VALUE FOR).

Привилегии на выполнение DDL операций

Выдача привилегий на создание, изменение или удаление объектов конкретного типа позволяет различным пользователям производить эти DDL операции.

- `CREATE` — разрешает создание объекта указанного типа метаданных;
- `ALTER ANY` — разрешает изменение любого объекта указанного типа метаданных;
- `DROP ANY` — разрешает удаление любого объекта указанного типа метаданных;
- `ALL [PRIVILEGES]` — объединяет привилегии `CREATE, ALTER` и `DROP` на указанный тип объекта.

DDL привилегии на базу данных

Оператор назначения привилегий на создание, удаление и изменение базы данных имеет несколько отличную форму от оператора назначения DDL привилегий на другие объекты метаданных.

- `CREATE` — разрешает создание базы данных;
- `ALTER` — разрешает изменение текущей базы данных;
- `DROP` — разрешает удаление текущей базы данных;
- `ALL [PRIVILEGES]` — объединяет привилегии `ALTER` и `DROP` на указанный тип объекта.

Привилегия `CREATE DATABASE` является особым видом привилегий, поскольку она сохраняется в базе данных безопасности. Список пользователей имеющих привилегию `CREATE DATABASE` можно посмотреть в виртуальной таблице `SEC$DB_CREATORS`. Привилегию на создание новой базы данных могут выдавать только администраторы в базе данных безопасности.

Привилегии `ALTER DATABASE` и `DROP DATABASE` относятся только к текущей базе данных. Привилегии на изменение и удаление текущей базы данных могут выдавать только администраторы.

Назначение ролей

Оператор `GRANT` может быть использован для назначения ролей для группы перечисленных пользователей. В этом случае после предложения `GRANT` следует список ролей, которые будут назначены списку пользователей, указанному после предложения `TO`.

При назначении роли пользователям можно указать предложение `WITH ADMIN OPTION`, которое дает право соответствующим пользователям назначать эти роли другим пользователям.

Если указано необязательное предложение `DEFAULT`, то пользователь при подключении к серверу получает все привилегии этой роли, какую бы роль он не указал при входе еще. Поэтому если пользователь не указывает роль при подключении к серверу, то он получает права только тех ролей, которые ему назначены с `DEFAULT`.

```
CREATE DATABASE 'LOCALHOST:/TMP/CUMROLES.FDB';
CREATE TABLE T(I INTEGER);
CREATE ROLE TINS;
CREATE ROLE CUMR;
GRANT INSERT ON T TO TINS;
GRANT SELECT ON T TO CUMR;
GRANT DEFAULT TINS TO USER1;
GRANT CUMR TO USER1;
```

```
CONNECT 'LOCALHOST:/TMP/CUMROLES.FDB' USER 'USER1' PASSWORD 'PAS' ROLE 'CUMR';
INSERT INTO T VALUES (1);
SELECT * FROM T;
```

Данный сценарий не выдаст ошибки.

Назначение политик

После создания политики безопасности (см. «Руководство администратора») ее можно назначить конкретному пользователю.

После назначения пользователям политик, касающихся требований к паролям, администратор должен сменить этим пользователям пароли, чтобы они удовлетворяли требованиям сопоставленных этим пользователям политик безопасности.

Политики назначаются только пользователям, но не ролям. Назначить политику несуществующему пользователю нельзя. У пользователя может быть только одна политика. Таким образом, чтобы отменить предыдущую политику и назначить новую, нужно просто еще раз выполнить оператор `GRANT POLICY <новая_политика>`.

Вновь созданным пользователям соответствует политика безопасности по умолчанию – `DEFAULT`. В ней отсутствуют какие-либо требования к паролям или сессиям пользователей. То есть для того, чтобы отменить требования политики для определенного пользователя, ему необходимо назначить политику по умолчанию:

```
GRANT POLICY "DEFAULT" TO <имя_пользователя>;
```

Выдача прав системным привилегиям

Благодаря поддержки системных привилегий в ядре, становится очень удобно предоставлять некоторые дополнительные привилегии пользователям уже имеющим какую-то системную привилегию. Для этих целей существует возможность использовать в качестве грантополучателя одну или несколько системных привилегий.

Описание системных привилегий можно посмотреть в разделе 26.2.

Пример.

Следующий оператор назначит все привилегии на представление `PLG$SRP_VIEW`, используемое в плагине управления пользователями `SRP`, системной привилегии `USER_MANAGEMENT`.

```
GRANT ALL ON PLG$SRP_VIEW TO SYSTEM PRIVILEGE USER_MANAGEMENT;
```

26.5.2 REVOKE

Оператор позволяет отозвать привилегии у пользователей, ролей, хранимых процедур, хранимых функций, пакетов, триггеров и представлений, которые были предоставлены оператором `GRANT`. Синтаксис оператора `REVOKE`:

Листинг 26.15. Синтаксис оператора `REVOKE`

```
REVOKE [GRANT OPTION FOR] <табличные привилегии>
   ON [TABLE] {<имя таблицы> | <имя представления>}
   FROM <список обладателей привилегий>
   [{GRANTED BY|AS} [USER] <имя грантора>]

REVOKE [GRANT OPTION FOR] EXECUTE ON {PROCEDURE | FUNCTION | PACKAGE} <имя процедуры/
функции/пакета>
   FROM <список обладателей привилегий>
   [{GRANTED BY|AS} [USER] <имя грантора>]

REVOKE [GRANT OPTION FOR] USAGE ON {EXCEPTION <имя искл-я>| {GENERATOR | SEQUENCE}
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
<имя генератора>}
   FROM <список обладателей привилегий>
   [{GRANTED BY|AS} [USER] <имя грантора>]

REVOKE [GRANT OPTION FOR] {ALL [PRIVILEGES] | {CREATE|ALTER ANY|DROP ANY} [,
{CREATE|ALTER ANY|DROP ANY}...] } <объект>
   FROM <список обладателей привилегий>
   [{GRANTED BY|AS} [USER] <имя грантора>]

REVOKE CREATE DATABASE FROM <пользователь обладатель> {,<пользователь обладатель>}

REVOKE [GRANT OPTION FOR] {ALL [PRIVILEGES]|{ALTER|DROP}[,{ALTER|DROP}]} DATABASE
   FROM <список обладателей привилегий>
   [{GRANTED BY|AS} [USER] <имя грантора>]

REVOKE [ADMIN OPTION FOR] [DEFAULT] <имя роли> [, [DEFAULT] <имя роли> ...]
   FROM [USER]|[ROLE] <имя польз-я/роли> [, [USER]|[ROLE] <имя польз-я/роли>...]
   [{GRANTED BY|AS} [USER] <имя грантора>]

REVOKE ALL ON ALL FROM <список обладателей привилегий>

<табличные привилегии> ::= ALL [PRIVILEGES] | <привилегия> [, <привилегия>...]

<привилегия> ::=  SELECT [(<имя столбца [, <имя столбца>...]>)]
                | DELETE
                | INSERT
                | UPDATE [(<имя столбца [, <имя столбца>...]>)]
                | REFERENCES [(<имя столбца [, <имя столбца>...]>)] }

<объект> ::=  TABLE
            | TABLESPACE
            | VIEW
            | PROCEDURE
            | FUNCTION
            | PACKAGE
            | GENERATOR
            | SEQUENCE
            | DOMAIN
            | EXCEPTION
            | ROLE
            | CHARACTER SET
            | COLLATION
            | FILTER
            | JOB

<список обладателей привилегий> ::={<объект обладатель>|<пользователь обладатель>}
                                 [, {<объект обладатель>|<пользователь обладатель>}...]

<объект обладатель> ::=  PROCEDURE <имя процедуры>
                       | FUNCTION <имя функции>
                       | PACKAGE <имя пакета>
                       | TRIGGER <имя триггера>
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
                          | VIEW <имя представления>
                          | SYSTEM PRIVILEGE <системная привилегия>

<пользователь обладатель> ::=  [USER] <имя пользователя>
                             | [ROLE] <имя роли>
                             | GROUP <имя группы в Unix>
```

Только пользователь, который назначил привилегию, может удалить её.

Оператор позволяет выполнить одно из следующих действий:

- отнять одну или несколько привилегий для таблиц и представлений у пользователей, ролей, хранимых процедур, хранимых функций, пакетов, триггеров и представлений;
- отнять права на выполнение процедуры, функции или пакета у пользователей, ролей, хранимых процедур, хранимых функций, пакетов, триггеров и представлений;
- отнять привилегии на контроль доступа к исключениям, последовательностям и генераторам;
- отнять одну или несколько привилегий на выполнение DDL операций над основными объектами базы данных у пользователей, ролей, хранимых процедур, хранимых функций, пакетов, триггеров и представлений;
- отнять привилегии на создание, удаление и изменение базы данных у пользователей, ролей, хранимых процедур, хранимых функций, пакетов, триггеров и представлений;
- отменить назначенные оператором `GRANT` роли;
- отменить все привелегии на всех объектах у пользователей.

Предложение FROM

В предложении `FROM` указывается список пользователей, ролей и объектов базы данных (процедур, функций, пакетов, триггеров и представлений), у которых будут отняты перечисленные привилегии. Необязательные предложения `USER` и `ROLE` позволяют уточнить, у кого именно отбирается привилегия. Если ключевое слово `USER` или `ROLE` не указано, то сервер проверяет, существует ли роль с данным именем, если таковой не существует, то привилегии отбираются у пользователя. Существование пользователя, у которого отбираются права, не проверяются при выполнении оператора `REVOKE`. Если привилегия отбирается у объекта базы данных, то необходимо обязательно указывать тип объекта.

GRANT OPTION FOR

Предложение `GRANT OPTION FOR` позволяет отменить для соответствующего пользователя или роли право предоставления другим пользователям или ролям привилегии к таблицам, представлениям, ролям, триггерам, хранимым процедурам.

ADMIN OPTION FOR

Предложение `ADMIN OPTION FOR` отменяет ранее предоставленную опцию — право на передачу предоставленной пользователю роли другим, не отменяя прав на роль.

GRANTED BY

Используя предложение `GRANTED BY` можно отозвать привилегии не от имени текущего пользователя, а от другого пользователя. Данное предложение доступно не всем пользователям (см. GRANT).

ALL ON ALL

Если указано предложение `ALL ON ALL`, то это позволяет отменить все привилегии (в том числе и роли) на всех объектах у пользователей и/или ролей. Это позволяет быстро заблокировать пользователю доступ к базе данных.

> Когда оператор `REVOKE ALL ON ALL` вызывается привилегированным пользователем (владельцем базы данных, `SYSDBA` или любым пользователем, с ролью `RDB$ADMIN`), удаляются все права независимо от того, кто их предоставил. В противном случае удаляются только права, предоставленные текущим пользователем.

Подробное описание различных типов привилегий см. в разделе 26.5.1.

Глава 27

Сессионное окружение

В этой главе описываются операторы управления сессионным окружением. Данные SQL операторы работают вне механизма управления транзакциями, изменения выполненные ими вступаю в силу немедленно.

Операторы управления сессионным окружением доступны в том числе и в PSQL коде. Это особенно полезно в `ON CONNECT` триггерах.

Операторы управления сессионным окружением разбиты на следующие группы:

- управления тайм-аутами;
- управление пулом внешних соединений;
- изменение текущей роли;
- управление обработкой типа `DECFLOAT`;
- управление часовым поясом;
- сброс сессионного окружения.

27.1 Тайм-ауты

В РЕД Базе Данных существует два вида тайм-аута: тайм-аут простоя соединения и тайм-аут выполнения SQL оператора.

27.1.1 Тайм-аут выполнения SQL оператора

Данная функциональность позволяет автоматически прекратить выполнение SQL оператора, если он выполняется дольше заданного значения тайм-аута.

Данная функция может быть полезна для:

- Администраторов баз данных, которые получают инструмент для ограничения времени выполнения тяжёлых запросов, которые потребляют много ресурсов;
- Разработчиков приложений, которые могут использовать тайм-ауты SQL операторов при написании и отладке сложных запросов с заранее неизвестным временем выполнения;
- Тестеров, которые могут использовать тайм-ауты SQL операторов для обнаружения долго выполняющихся запросов и обеспечения конечного времени выполнения набора тестов.

Эта функциональность работает следующим образом. Когда начинается выполнение оператора (или открывается курсор) РЕД База Данных запускает специальный таймер. Выборка записей (`fetch`) не сбрасывает таймер. Таймер останавливается, если выполнение SQL оператора закончено или извлечена (`fetch`) последняя запись.

По истечению тайм-аута:

- Если выполнение SQL оператора активно, оно останавливается в заданный момент.
- Если SQL оператор не активен в данный момент (например между выборками (`fetch`)), то он будет помечен как отменённый, следующая выборка (`fetch`) прервёт выполнение и будет возвращена ошибка.

Значение тайм-аута может быть установлено:

- На уровне базы данных. Значение параметра `StatementTimeout` может быть установлено в `firebird.conf` (или `databases.conf`) администратором базы данных. Область действия — все операторы во всех соединениях. Параметр `StatementTimeout` устанавливает тайм-аут в секундах, по истечении которого выполнение SQL операторов будет отменено. Ноль означает, что тайм-аут не установлен. Значение по умолчанию равно 0.

- На уровне соединения. Может быть установлен с использованием API (в миллисекундах) или с помощью SQL оператора `SET STATEMENT TIMEOUT`. Область действия текущее подключение.

```
SET STATEMENT TIMEOUT <значение> [HOUR | MINUTE | SECOND | MILLISECOND]
```

  В качестве параметра выступает значение тайм-аута выполнения SQL операторов в указанных единицах измерения времени. Если единица измерения времени не указано, то по умолчанию значение тайм-аута измеряется в секундах.
- На уровне оператора. Может быть установлен с использованием API (в миллисекундах). Область действия — текущий SQL оператор.

Эффективное значение тайм-аута SQL оператора вычисляется каждый раз, когда запускается SQL оператор (открывается курсор), следующим образом:

- если тайм-аут не установлен на уровне оператора, будет использовано значение тайм-аута уровня соединения;
- если тайм-аут не установлен на уровне соединения, будет использовано значение тайм-аута уровня базы данных;
- значение тайм-аута не может быть больше, чем значение установленное на уровне базы данных. Таким образом, значение тайм-аута может перекрываться разработчиком приложения в более низких областях, но оно не может выти за пределы установленные DBA в конфигурации.

Нулевой тайм-аут не обозначает отсутствие тайм-аута, просто в этом случае таймер выполнения оператора не запускается.

Несмотря на то, что тайм-аут выполнения SQL оператора может быть установлен в миллисекундах, абсолютная точность не гарантируется. При высокой нагрузке он может быть менее точным. Единственная гарантия которую может дать РЕД База Данных это то, что тайм-аут не сработает раньше указанного момента. Клиентское приложение может ждать больше времени, чем установленное значение тайм-аута, если движку РЕД Базы Данных необходимо отменить множество действий связанных с отменой оператора.

Тайм-аут выполнения оператора игнорируется для всех внутренних запросов, которые использует движком РЕД Базы Данных. Кроме того, тайм-аут игнорируется для DDL операторов.

27.1.2 Тайм-аут простоя соединения

Данная функциональность позволяет автоматически закрывать пользовательские подключения после периода бездействия. Она может быть использована администраторами баз данных, чтобы принудительно закрывать старые неактивные соединения и освобождать связанные с ними ресурсы. Приложения и инструменты разработчика могут использовать её как замену самодельного контроля за временем жизни подключения.

Рекомендуется (но не обязательно) устанавливать тай-аут простоя в разумное большое значение, например, несколько часов. По умолчанию эта функция отключена.

Эта функциональность работает следующим образом. Когда пользовательский вызов API покидает движок, запускается специальный таймер связанный с текущим подключением. Как только пользовательский вызов входит в движок, таймер ожидания останавливается. Если тайм-аут простоя истечёт движок закроет соединение так как будто произошло асинхронная отмена подключения:

- все активные операторы и курсоры закрываются;
- все активные транзакции откатываются;
- сетевые соединения не закрываются в данный момент. Это позволяет клиентскому приложение получить точный код ошибки при следующем вызове API. Сетевое соединение будет закрыто на стороне сервера после того, как ошибка сообщена, или если клиентская сторона отключится по истечению тайм-аута сети.

Тайм-аут простоя соединения может быть установлен:

- На уровне базы данных. Значение параметра `ConnectionIdleTimeout` может быть установлено в `firebird.conf` (или `databases.conf`) администратором базы данных. Область действия — все пользовательские подключения, исключая системные подключения (`garbage collector`, `cache writer`, и др.). Параметр `ConnectionIdleTimeout` устанавливает тайм-аут в минутах, по истечении которого неактивное соединение будет разорвано движком. Ноль означает, что тайм-аут не установлен. Значение по умолчанию равно 0.
- На уровне подключения. Может быть установлен с использованием API (в секундах) или с помощью SQL оператора `SET SESSION IDLE TIMEOUT`. Область действия — все операторы в текущем подключении.

  ```
  SET SESSION IDLE TIMEOUT <значение> [HOUR | MINUTE | SECOND]
  ```

  В качестве параметра выступает значение тайм-аута простоя в указанных единицах измерения времени. Если единица измерения времени не указано, то по умолчанию значение тайм-аута измеряется в минутах.

Эффективное значение тайм-аута простоя вычисляется каждый раз, когда пользовательский вызов API покидает движок, следующим образом:

- если тайм-аут не установлен на уровне подключения, будет использовано значение уровня базы данных;
- значение тайм-аута не может быть больше, чем значение установленное на уровне базы данных. Таким образом, значение тайм-аута простоя может перекрываться разработчиком приложения для заданного подключения, но оно не может выти за пределы установленные DBA в конфигурации.

Нулевой тайм-аут не обозначает отсутствие тайм-аута, просто в этом случае таймер ожидания не запускается.

Несмотря на то, что тайм-аут простоя может быть установлен в секундах, абсолютная точность не гарантируется. При высокой нагрузке он может быть менее точным. Единственная гарантия которую может дать РЕД База Данных это то, что тайм-аут не сработает раньше указанного момента.

27.2 Пул внешних соединений

Каждое внешнее соединение (созданное оператором `EXECUTE STATEMENT ... ON EXTERNAL`) при создании связывается с пулом соединений (подробнее см. `раздел %s`). Данная группа операторов позволяет управлять пулом внешних соединений:

- `ALTER EXTERNAL CONNECTIONS POOL SET SIZE <размер>` — устанавливает максимальное количество бездействующих соединений;
- `ALTER EXTERNAL CONNECTIONS POOL SET LIFETIME <значение> {SECOND | MINUTE | HOUR}` — устанавливает время жизни бездействующих соединений;
- `ALTER EXTERNAL CONNECTIONS POOL CLEAR ALL` — закрывает все бездействующие соединения;
- `ALTER EXTERNAL CONNECTIONS POOL CLEAR OLDEST` — закрывает бездействующие соединения у которых истекло время жизни.

При его подготовке они описываются как DDL операторы, но имеют немедленный эффект: то есть они выполняются немедленно и полностью, не дожидаясь фиксации транзакции. Изменения применяются к экземпляру пула в памяти в текущем процессе РЕД Базы Данных. Поэтому изменение в одном классическом процессе не влияет на другие классические процессы. Изменения не являются постоянными и после перезапуска сервера будет использовать настройки пула из `firebird.conf`.

Для выполнения операторов данной группы требуется системная привилегия `MODIFY_EXT_CONN_POOL` (подробнее о системных привелегиях см. `CREATE ROLE`).

Подробнее см. эти операторы в Приложение Ж.

27.3 Изменение текущей роли

Данная группа операторов позволяет изменять текущую роль:

- `SET ROLE <имя роль>` — изменение текущей роли;
- `SET TRUSTED ROLE` — включает доступ доверенной роли, при условии, что `CURRENT_USER` получен с помощью доверительной аутентификации и роль доступна.

Подробнее см. эти операторы в Приложение Ж.

27.4 Управление часовым поясом и обработкой часовых поясов

Данная группа операторов позволяет управлять часовым поясом:

- `SET TIME ZONE` — изменение часового пояса сеанса;
- `SET TIME ZONE BIND {NATIVE | LEGACY}` — изменение привязки типов с `WITH TIME ZONE`.

Подробнее см. эти операторы в Приложение Ж.

27.5 Управление обработкой DECFLOAT

Для изменения режима округления для типа `DECFLOAT` создан оператор `SET DECFLOAT ROUND <режим>`.

Поддерживаются следующие режимы округления совместимые со стандартом IEEE: `CEILING, UP, HALF_UP, HALF_EVEN, HALF_DOWN, DOWN, FLOOR, REROUND`.

Подробнее см. эти операторы в Приложение Ж.

27.6 Сброс сессионного окружения

Оператор `ALTER SESSION RESET` сбрасывает сеансовое окружение (подключения) к исходному состоянию. Эта функциональность полезна, если сеанс используется повторно, вместо того, чтобы производить отключение/подключение.

```
ALTER SESSION RESET
```

Данный оператор делает следующее:

- сбрасывает установленные параметры `DECFLOAT (BIND, TRAP` и `ROUND)` в значения по умолчанию;
- сбрасывает тайм-ауты сессии и оператора в 0;
- удаляет все контекстные переменные из пространства имён `USER_SESSION`;
- очищает содержимое всех используемых глобальных таблиц уровня соединения (`COMMIT PRESERVE ROWS`);
- сбрасывает роль в значение переданное в DPB (указанное при подключении) и очищает кэш привелегий (если роль была изменена с помощью оператора `SET ROLE`);
- текущая активная транзакция откатывается и стартуется новая с теми же параметрами, после рестарта.

Если в текущей активной транзакции были произведены изменения, то будет выдано предупреждение.

Если в текущей сессии активны другие транзакции, то будет выдана ошибка. При проверке транзакций перед сбросом сессии подготовленные 2PC транзакции игнорируются.

Приложение А Зарезервированные и ключевые слова

Зарезервированные слова нельзя использовать в качестве имен объектов базы данных, переменных и параметров. Список зарезервированных слов представлен в таблице А.1. Не рекомендуется использовать и ключевые слова как имена объектов базы данных, поскольку не исключена вероятность перевода их в разряд зарезервированных в следующих версиях системы управления базами данных.

Таблица А.1 — Список зарезервированных слов SQL РЕД База Данных

ADD	ADMIN	ALL
ALTER	AND	ANY
AS	AT	AVG
BEGIN	BETWEEN	BIGINT
BINARY	BIT_LENGTH	BLOB
BOOLEAN	BOTH	BY
CALL	CASE	CAST
CHAR	CHARACTER	CHARACTER_LENGTH
CHAR_LENGTH	CHECK	CLOSE
COLLATE	COLUMN	COMMENT
COMMIT	CONNECT	CONSTRAINT
CORR	COUNT	COVAR_POP
COVAR_SAMP	CREATE	CROSS
CURRENT	CURRENT_DATE	CURRENT_ROLE
CURRENT_CONNECTION	CURRENT_TIME	CURRENT_TIMESTAMP
CURRENT_TRANSACTION	CURRENT_USER	CURSOR
DATE	DAY	DEC
DECFLOAT	DECIMAL	DECLARE
DEFAULT	DELETE	DELETING
DETERMINISTIC	DISCONNECT	DISTINCT
DOUBLE	DROP	ELSE
END	ESCAPE	EXECUTE
EXISTS	EXTERNAL	EXTRACT
FALSE	FETCH	FILTER
FLOAT	FOR	FOREIGN
FROM	FULL	FUNCTION
GDSCODE	GLOBAL	GRANT
GROUP	HAVING	HOUR
IN	INDEX	INNER
INSENSITIVE	INSERT	INSERTING
INT	INT128	INTEGER

(разрыв таблицы)

(разрыв таблицы)

INTO	IS	JOIN
LATERAL	LEADING	LEFT
LIKE	LOCAL	LOCALTIME
LOCALTIMESTAMP	LONG	LOWER
MAX	MERGE	MIN
MINUTE	MONTH	NATIONAL
NATURAL	NCHAR	NO
NOT	NULL	NUMERIC
OCTET_LENGTH	OF	OFFSET
ON	ONLY	OPEN
OR	ORDER	OUTER
OVER	PARAMETER	PLAN
POSITION	POST_EVENT	PRECISION
PRIMARY	PROCEDURE	PUBLICATION
RDB$DB_KEY	RDB$ERROR	RDB$GET_CONTEXT
RDB$RECORD_VERSION	RDB$ROLE_IN_USE	RDB$SET_CONTEXT
RDB$SYSTEM_PRIVILEGE	REAL	RECORD_VERSION
RECREATE	RECURSIVE	RDB$GET_TRANSACTION_CN
REFERENCES	REGR_AVGX	REGR_AVGY
REGR_COUNT	REGR_INTERCEPT	REGR_R2
REGR_SLOPE	REGR_SXX	REGR_SXY
REGR_SYY	RELEASE	RESETTING
RETURN	RETURNING_VALUES	RETURNS
REVOKE	RIGHT	ROLLBACK
ROW	ROWS	ROW_COUNT
SAVEPOINT	SCROLL	SECOND
SELECT	SENSITIVE	SET
SIMILAR	SMALLINT	SOME
SQLCODE	SQLSTATE	START
STDDEV_POP	STDDEV_SAMP	SUM
TABLE	THEN	TIME
TIMESTAMP	TIMEZONE_HOUR	TIMEZONE_MINUTE
TO	TRAILING	TRIGGER
TRIM	TRUE	UNBOUNDED
UNION	UNIQUE	UNKNOWN
UPDATE	UPDATING	UPPER
USER	USING	VALUE
VALUES	VARBINARY	VARCHAR

(разрыв таблицы)

(разрыв таблицы)

VARIABLE	VARYING	VAR_POP
VAR_SAMP	VIEW	WHEN
WHERE	WHILE	WINDOW
WITH	WITHOUT	YEAR

В таблице A.2 представлен список ключевых слов. Это слова, используемые в операторах SQL, но которые в принципе можно использовать в качестве имен объектов базы данных.

Таблица A.2 — Список ключевых слов SQL РЕД База Данных

!<	!=	!>
(	)	,
<	<=	<>
=	>	>=
:=	^<	^=
^>	||	~<
~=	~>	ABS
ABSOLUTE	ACCENT	ACOS
ACOSH	ACTION	ACTIVE
ADD	ADMIN	AFTER
ALL	ALTER	ALWAYS
AND	ANY	AS
ASC	ASCENDING	ASCII_CHAR
ASCII_VAL	ASIN	ASINH
AT	ATAN	ATAN2
ATANH	AUTO	AUTONOMOUS
AVG	BACKUP	BASE64_DECODE
BASE64_ENCODE	BEFORE	BEGIN
BETWEEN	BIGINT	BINARY
BIND	BIN_AND	BIN_NOT
BIN_OR	BIN_SHL	BIN_SHR
BIN_XOR	BIT_LENGTH	BLOB
BLOCK	BODY	BOOLEAN
BOTH	BREAK	BY
CALLER	CASCADE	CASE
CAST	CEIL	CEILING
CHAR	CHARACTER	CHARACTER_LENGTH
CHAR_LENGTH	CHAR_TO_UUID	CHECK
CLEAR	CLOSE	COALESCE
COLLATE	COLLATION	COLUMN

(разрыв таблицы)

(разрыв таблицы)

COMMENT	COMMIT	COMMITTED
COMMON	COMPARE_DECFLOAT	COMPUTED
CONDITIONAL	CONNECT	CONNECTIONS
CONSISTENCY	CONSTRAINT	CONTAINING
CONTINUE	CORR	COS
COSH	COT	COUNT
COUNTER	COVAR_POP	COVAR_SAMP
CREATE	CROSS	CRYPT_HASH
CSTRING	CTR_BIG_ENDIAN	CTR_LENGTH
CTR_LITTLE_ENDIAN	CUME_DIST	CURRENT
CURRENT_CONNECTION	CURRENT_DATE	CURRENT_ROLE
CURRENT_TIME	CURRENT_TIMESTAMP	CURRENT_TRANSACTION
CURRENT_USER	CURSOR	DATA
DATABASE	DATE	DATEADD
DATEDIFF	DAY	DDL
DEBUG	DEC	DECFLOAT
DECIMAL	DECLARE	DECODE
DECRYPT	DEFAULT	DEFINER
DELETE	DELETING	DENSE_RANK
DESC	DESCENDING	DESCRIPTOR
DETERMINISTIC	DIFFERENCE	DISABLE
DISCONNECT	DISTINCT	DO
DOMAIN	DOUBLE	DROP
ELSE	ENABLE	ENCRYPT
END	ENGINE	ENTRY_POINT
ESCAPE	EXCEPTION	EXCESS
EXCLUDE	EXECUTE	EXISTS
EXIT	EXP	EXTENDED
EXTERNAL	EXTRACT	FALSE
FETCH	FILE	FILTER
FIRST	FIRSTNAME	FIRST_DAY
FIRST_VALUE	FLOAT	FLOOR
FOLLOWING	FOR	FOREIGN
FREE_IT	FROM	FULL
FUNCTION	GDSCODE	GENERATED
GENERATOR	GEN_ID	GEN_UUID
GLOBAL	GRANT	GRANTED
GROUP	HASH	HAVING

(разрыв таблицы)

(разрыв таблицы)

HEX_DECODE	HEX_ENCODE	HOUR
IDENTITY	IDLE	IF
IGNORE	IIF	IN
INACTIVE	INCLUDE	INCREMENT
INDEX	INNER	INPUT_TYPE
INSENSITIVE	INSERT	INSERTING
INT	INT128	INTEGER
INTO	INVOKER	IS
ISOLATION	IV	JOIN
KEY	LAG	LAST
LASTNAME	LAST_DAY	LAST_VALUE
LATERAL	LEAD	LEADING
LEAVE	LEFT	LEGACY
LENGTH	LEVEL	LIFETIME
LIKE	LIMBO	LINGER
LIST	LN	LOCAL
LOCALTIME	LOCALTIMESTAMP	LOCK
LOG	LOG10	LONG
LOWER	LPAD	LPARAM
MAKE_DBKEY	MANUAL	MAPPING
MATCHED	MATCHING	MAX
MAXVALUE	MERGE	MESSAGE
MIDDLENAME	MILLISECOND	MIN
MINUTE	MINVALUE	MOD
MODE	MODULE_NAME	MONTH
NAME	NAMES	NATIONAL
NATIVE	NATURAL	NCHAR
NEXT	NO	NORMALIZE_DECFLOAT
NOT	NTH_VALUE	NTILE
NULL	NULLIF	NULLS
NUMBER	NUMERIC	OCTET_LENGTH
OF	OFFSET	OLDEST
ON	ONLY	OPEN
OPTION	OR	ORDER
OS_NAME	OTHERS	OUTER
OUTPUT_TYPE	OVER	OVERFLOW
OVERLAY	OVERRIDING	PACKAGE
PAD	PAGE	PAGES

(разрыв таблицы)

(разрыв таблицы)

PAGE_SIZE	PARAMETER	PARTITION
PASSWORD	PERCENT_RANK	PI
PKCS_1_5	PLACING	PLAN
PLUGIN	POOL	POSITION
POST_EVENT	POWER	PRECEDING
PRECISION	PRESERVE	PRIMARY
PRIOR	PRIVILEGE	PRIVILEGES
PROCEDURE	PROTECTED	PUBLICATION
QUANTIZE	RAND	RANGE
RANK	RDB$DB_KEY	RDB$ERROR
RDB$GET_CONTEXT	RDB$GET_TRANSACTION_CN	RDB$RECORD_VERSION
RDB$ROLE_IN_USE	RDB$SET_CONTEXT	RDB$SYSTEM_PRIVILEGE
READ	REAL	RECORD_VERSION
RECREATE	RECURSIVE	REFERENCES
REGR_AVGX	REGR_AVGY	REGR_COUNT
REGR_INTERCEPT	REGR_R2	REGR_SLOPE
REGR_SXX	REGR_SXY	REGR_SYY
RELATIVE	RELEASE	REPLACE
REQUESTS	RESERV	RESERVING
RESET	RESETTING	RESTART
RESTRICT	RETAIN	RETURN
RETURNING	RETURNING_VALUES	RETURNS
REVERSE	REVOKE	RIGHT
ROLE	ROLLBACK	ROUND
ROW	ROWS	ROW_COUNT
ROW_NUMBER	RPAD	RSA_DECRYPT
RSA_ENCRYPT	RSA_PRIVATE	RSA_PUBLIC
RSA_SIGN_HASH	RSA_VERIFY_HASH	SALT_LENGTH
SAVEPOINT	SCALAR_ARRAY	SCHEMA
SCROLL	SECOND	SECURITY
SEGMENT	SELECT	SENSITIVE
SEQUENCE	SERVERWIDE	SESSION
SET	SHADOW	SHARED
SIGN	SIGNATURE	SIMILAR
SIN	SINGULAR	SINH
SIZE	SKIP	SMALLINT
SNAPSHOT	SOME	SORT
SOURCE	SPACE	SQL

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	SQLSTATE	SQRT
STABILITY	START	STARTING
STARTS	STATEMENT	STATISTICS
STDDEV_POP	STDDEV_SAMP	SUBSTRING
SUB_TYPE	SUM	SUSPEND
SYSTEM	TABLE	TAGS
TAN	TANH	TEMPORARY
THEN	TIES	TIME
TIMEOUT	TIMESTAMP	TIMEZONE_HOUR
TIMEZONE_MINUTE	TO	TOTALORDER
TRAILING	TRANSACTION	TRAPS
TRIGGER	TRIM	TRUE
TRUNC	TRUSTED	TWO_PHASE
TYPE	UNBOUNDED	UNCOMMITTED
UNDO	UNION	UNIQUE
UNKNOWN	UPDATE	UPDATING
UPPER	USAGE	USER
USING	UUID_TO_CHAR	VALUE
VALUES	VARBINARY	VARCHAR
VARIABLE	VARYING	VAR_POP
VAR_SAMP	VIEW	WAIT
WEEK	WEEKDAY	WHEN
WHERE	WHILE	WINDOW
WITH	WITHOUT	WORK
WRITE	YEAR	YEARDAY
ZONE		

Приложение Б Коды ошибок РЕД База Данных

Для обработки ошибок, возникающих в процессе работы с базой данных, используются контекстные переменные SQLCODE, GDSCODE и SQLSTATE. Эти контекстные переменные могут применяться только в хранимых процедурах и триггерах в блоках обработки ошибок WHEN. За пределами таких блоков SQLCODE и GDSCODE имеют нулевое значение, SQLSTATE — равен '00000' (а вне PSQL не существует вообще).

Б.1 Коды ошибок GDSCODE и SQLCODE

Значения SQLCODE представлены в таблице Б.1.

Таблица Б.1 — Значения SQLCODE

SQLCODE	Смысл
<0	Произошла ошибка при попытке выполнения оператора. Действие не выполнено.
0	Нормальное завершение выполнения оператора.
1 ÷ 99	Системные предупреждения или информационные сообщения. Оператор выполнен.
+100	Достигнут конец набора данных.

Одному значению кода SQLCODE может соответствовать несколько вариантов ошибок и сообщений. Более детальную информацию об ошибке можно получить, используя значение контекстной переменной GDSCODE.

В настоящее время SQLCODE считаются устаревшим. В следующих версиях поддержка SQLCODE может полностью прекратиться

В таблице Б.2 приведены значения переменных SQLCODE и GDSCODE, а также тексты выдаваемых сервером базы данных сообщений об ошибках и перевод этих текстов на русский язык.

Таблица Б.2 — Значения кодов SQLCODE и GDSCODE

SQLCODE	GDSCODE	Символ	Текст сообщения
-802	335544321	arith_except	arithmetic exception, numeric overflow, or string truncation Арифметическое исключение, числовое переполнение или строковое обрезание.
-901	335544322	bad_dbkey	invalid database key Неверный ключ базы данных.
-922	335544323	bad_db_format	file @1 is not a valid database Файл @1 не является допустимой базой данных
-904	335544324	bad_db_handle	invalid database handle (no active connection) Неверный дескриптор базы данных (нет активного соединения)
-924	335544325	bad_dpb_content	bad parameters on attach or create database Неверные параметры при подключении или при создании базы данных.
-901	335544326	bad_dpb_form	unrecognized database parameter block Блок параметров базы данных не распознан.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	335544327	bad_req_handle	invalid request handle Неверный запрос дескриптора.
-901	335544328	bad_segstr_handle	invalid BLOB handle Неверный дескриптор BLOB.
-901	335544329	bad_segstr_id	invalid BLOB ID Неверный идентификатор BLOB.
-901	335544330	bad_tpb_content	invalid parameter in transaction parameter block Неверный параметр в блоке параметров транзакции.
-901	335544331	bad_tpb_form	invalid format for transaction parameter block Неверный формат блока параметров транзакции.
-901	335544332	bad_trans_handle	invalid transaction handle (expecting explicit transaction start) Неверный дескриптор транзакции (ожидается явный запуск транзакции).
-902	335544333	bug_check	internal Firebird consistency check (@1) Внутренняя проверка целостности программного обеспечения (@1)
-413	335544334	convert_error	conversion error from string "@1" Ошибка преобразования для строки "@1".
-902	335544335	db_corrupt	database file appears corrupt (@1) Файл базы данных является поврежденным (@1)
-913	335544336	deadlock	deadlock Взаимная блокировка.
-901	335544337	excess_trans	attempt to start more than @1 transactions Не используется.
100	335544338	from_no_match	no match for first value expression Нет соответствия для первого значения выражения.
-901	335544339	infinap	information type inappropriate for object specified Информационный тип не соответствует указанному объекту.
-901	335544340	infona	no information of this type available for object specified Не используется.
-901	335544341	infunk	unknown information item Неизвестный информационный элемент.
-901	335544342	integ_fail	action cancelled by trigger (@1) to preserve data integrity Действие отменено триггером (@1), чтобы сохранить целостность данных.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-104	335544343	invalid_blr	invalid request BLR at offset @1 Неверный запрос BLR со смещением @1.
-902	335544344	io_error	I/O error during "@1" operation for file "@2" Ошибка ввода - вывода во время операции @1 для файла @2
-901	335544345	lock_conflict	lock conflict on no wait transaction Конфликт блокировки для транзакции NO WAIT
-902	335544346	metadata_corrupt	corrupt system table Повреждение системной таблицы
-625	335544347	not_valid	validation error for column @1, value "@2" Ошибка проверки данных для столбца @1, значение "@2".
-508	335544348	no_cur_rec	no current record for fetch operation Нет текущей записи для операции FETCH.
-803	335544349	no_dup	attempt to store duplicate value (visible to active transactions) in unique index "@1" Попытка сохранить дубликат значения (видимые при активных транзакциях) в уникальном индексе "@1".
-901	335544350	no_finish	program attempted to exit without finishing database Не используется.
-607	335544351	no_meta_update	unsuccessful metadata update Неудачное обновление метаданных.
-551	335544352	no_priv	no permission for @1 access to @2 @3 Не существует полномочий для доступа @1 к @2 @3.
-901	335544353	no_recon	transaction is not in limbo Транзакция не является зависшей (limbo).
100	335544354	no_record	invalid database key Не используется.
-901	335544355	no_segstr_close	BLOB was not closed Не используется.
-820	335544356	obsolete_metadata	metadata is obsolete Не используется.
-901	335544357	open_trans	cannot disconnect database with open transactions (@1 active) Невозможно отключиться от базы данных при наличии открытой транзакции (активная транзакция @1).
-901	335544358	port_len	message length error (encountered @1, expected @2) Ошибка длины сообщения (встречено @1, ожидается @2)

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-151	335544359	read_only_field	attempted update of read-only column Попытка обновить доступный только для чтения столбец.
-150	335544360	read_only_rel	attempted update of read-only table Не используется.
-817	335544361	read_only_trans	attempted update during read-only transaction Попытка выполнить изменения во время выполнения транзакции только для чтения.
-150	335544362	read_only_view	cannot update read-only view @1 Невозможно изменить представление @1 только для чтения.
-901	335544363	req_no_trans	no transaction for request Для запроса нет транзакции.
-901	335544364	req_sync	request synchronization error Ошибка синхронизации запроса.
-901	335544365	req_wrong_db	request referenced an unavailable database Не используется.
101	335544366	segment	segment buffer length shorter than expected Длина сегмента буфера меньше, чем ожидается.
100	335544367	segstr_eof	attempted retrieval of more segments than exist Попытка обращения к сегменту большему, чем их существует
-402	335544368	segstr_no_op	attempted invalid operation on a BLOB Не используется.
-901	335544369	segstr_no_read	attempted read of a new, open BLOB Не используется.
-901	335544370	segstr_no_trans	attempted action on BLOB outside transaction Не используется.
-817	335544371	segstr_no_write	attempted write to read-only BLOB Попытка записи в тип данных BLOB только для чтения.
-901	335544372	segstr_wrong_db	attempted reference to BLOB in unavailable database Не используется.
-902	335544373	sys_request	operating system directive @1 failed Директива операционной системы @1 не удалась
-596	335544374	stream_eof	attempt to fetch past the last record in a record stream Попытка получения в потоке записей записи, следующей за последней.
-904	335544375	unavailable	unavailable database Недоступная база данных

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	335544376	unres_rel	table @1 was omitted from the transaction reserving list Не используется.
-901	335544377	uns_ext	request includes a DSRI extension not supported in this implementation Запрос включает расширение DSRI, не поддерживаемое в этой реализации.
-901	335544378	wish_list	feature is not supported Возможность не поддерживается.
-820	335544379	wrong_ods	unsupported on-disk structure for file @1; found @2.@3, support @4.@5 Неподдерживаемая ODS для файла @1, обнаружена @2.@3, поддерживается @4.@5.
-804	335544380	wronumarg	wrong number of arguments on call Не используется.
-904	335544381	imp_exc	Implementation limit exceeded Исчерпан лимит выполнения
-901	335544382	random	@1
-901	335544383	fatal_conflict	unrecoverable conflict with limbo transaction @1 Не используется.
-902	335544384	badblk	internal error Внутренняя ошибка.
-902	335544385	invpoolcl	internal error Не используется.
-904	335544386	nopoolids	too many requests Не используется.
-902	335544387	relbadblk	internal error Не используется.
-902	335544388	blktoobig	block size exceeds implementation restriction Размер блока превышает ограничение реализации
-904	335544389	bufexh	buffer exhausted Не используется.
-104	335544390	syntaxerr	BLR syntax error: expected @1 at offset @2, encountered @3 Ошибка синтаксиса BLR: ожидается @1 по смещению @2, встречено @3.
-904	335544391	bufinuse	buffer in use Не используется.
-901	335544392	bdbincon	internal error Не используется.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-904	335544393	reqinuse	request in use Запрос используется.
-902	335544394	badodsver	incompatible version of on-disk structure Не используется.
-219	335544395	relnotdef	table @1 is not defined Таблица @1 не определена.
-205	335544396	fldnotdef	column @1 is not defined in table @2 Столбец @1 не определен в таблице @2.
-902	335544397	dirtypage	internal error Не используется.
-902	335544398	waifortra	internal error Не используется.
-902	335544399	doubleloc	internal error Не используется.
-902	335544400	nodnotfnd	internal error Не используется.
-902	335544401	dupnodfnd	internal error Не используется.
-902	335544402	locnotmar	internal error Не используется.
-689	335544403	badpagtyp	page @1 is of wrong type (expected @2, found @3) Страница @1 имеет неверный тип (ожидается @2, обнаружена @3).
-902	335544404	corrupt	database corrupted База данных повреждена.
-902	335544405	badpage	checksum error on database page @1 Не используется.
-902	335544406	badindex	index is broken Не используется.
-901	335544407	dbbnotzer	database handle not zero Не используется.
-901	335544408	tranotzer	transaction handle not zero Не используется.
-902	335544409	trareqmis	transaction-request mismatch (synchronization error) Транзакция - несогласованный запрос (ошибка синхронизации)
-902	335544410	badhndcnt	bad handle count Не используется.
-902	335544411	wrotpbver	wrong version of transaction parameter block Неверная версия блока параметров транзакции

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-902	335544412	wroblrver	unsupported BLR version (expected @1, encountered @2) Не поддерживаемая версия BLR (ожидается @1 встретилась @2)
-902	335544413	wrodpbver	wrong version of database parameter block Неверная версия блока параметров базы данных.
-402	335544414	blobnotsup	BLOB and array data types are not supported for @1 operation Типы данных BLOB и массив не поддерживаются для операции @1.
-902	335544415	badrelation	database corrupted Не используется.
-902	335544416	nodetach	internal error Не используется.
-902	335544417	notremote	internal error Не используется.
-901	335544418	trainlim	transaction in limbo Зависшая транзакция.
-901	335544419	notinlim	transaction not in limbo Не используется.
-901	335544420	traoutsta	transaction outstanding Не используется.
-923	335544421	connect_reject	connection rejected by remote interface Соединение отменено удаленным интерфейсом
-902	335544422	dbfile	internal error Внутренняя ошибка.
-902	335544423	orphan	internal error Внутренняя ошибка.
-904	335544424	no_lock_mgr	no lock manager available Не используется.
-104	335544425	ctxinuse	context already in use (BLR error) Контекст находится в использовании (ошибка BLR).
-104	335544426	ctxnotdef	context not defined (BLR error) Контекст не определен (ошибка BLR).
-402	335544427	datnotsup	data operation not supported Операция данных не поддерживается.
-901	335544428	badmsgnum	undefined message number Неопределенный номер сообщения
-104	335544429	badparnum	undefined parameter number Неверный номер параметра.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-904	335544430	virmemexh	unable to allocate memory from operating system Невозможно выделить память в операционной системе.
-901	335544431	blocking_signal	blocking signal has been received Не используется.
-902	335544432	lockmanerr	lock manager error Ошибка менеджера блокировок.
-924	335544433	journerr	communication error with journal "@1" Не используется.
-664	335544434	keytoobig	key size exceeds implementation restriction for index "@1" Размер ключа превышает ограничения реализации для индекса "@1"
-407	335544435	nullsegkey	null segment of UNIQUE KEY Не используется.
-902	335544436	sqlerr	SQL error code = @1 Код SQL ошибки = @1
-820	335544437	wrodynver	wrong DYN version Не используется.
-172	335544438	funnotdef	function @1 is not defined Функция @1 не определена.
-171	335544439	funmismat	function @1 could not be matched Функции @1 нельзя найти соответствие.
-104	335544440	bad_msg_vec	Не используется.
-924	335544441	bad_detach	database detach completed with errors Не используется.
-901	335544442	noargacc_read	database system cannot read argument @1 Не используется.
-901	335544443	noargacc_write	database system cannot write argument @1 Не используется.
-817	335544444	read_only	operation not supported Операция не поддерживается.
-677	335544445	ext_err	@1 extension error Не используется.
-150	335544446	non_updatable	not updatable Не используется.
-926	335544447	no_rollback	no rollback performed Не используется.
-902	335544448	bad_sec_info	Не используется.
-902	335544449	invalid_sec_info	Не используется.
-901	335544450	misc_interpreted	@1

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-904	335544451	update_conflict	update conflicts with concurrent update Изменение конфликтует с конкурирующим изменением.
-906	335544452	unlicensed	product @1 is not licensed Продукт @ 1 не лицензирован.
-904	335544453	obj_in_use	object @1 is in use Объект @1 используется.
-413	335544454	nofilter	filter not found to convert type @1 to type @2 Не найден фильтр для преобразования типа @1 в тип @2 (для BLOB).
-904	335544455	shadow_accessed	cannot attach active shadow file Невозможно соединиться с активным файлом теневой копии
-104	335544456	invalid_sdl	invalid slice description language at offset @1 Неверный фрагмент языка описания по смещению @1.
-406	335544457	out_of_bounds	subscript out of bounds Выход за пределы диапазона.
-171	335544458	invalid_dimension	column not array or invalid dimensions (expected @1, encountered @2) Столбец не является массивом или неверная размерность(ожидается @1, встретилась @2).
-911	335544459	rec_in_limbo	record from transaction @1 is stuck in limbo Запись транзакции @1 становится зависшей
-904	335544460	shadow_missing	a file in manual shadow @1 is unavailable Файл в ручной теневой копии @1 недоступен
-923	335544461	cant_validate	secondary server attachments cannot validate databases Вторичные подключения к серверу не могут проверять базы данных.
-923	335544462	cant_start_journal	secondary server attachments cannot start journaling Вторичные подключения к серверу не могут начать журналирование.
-204	335544463	gennotdef	generator @1 is not defined Генератор @1 не определен.
-923	335544464	cant_start_logging	secondary server attachments cannot start logging Не используется.
-685	335544465	bad_segstr_type	invalid BLOB type for operation Неверный тип BLOB для операции.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-530	335544466	foreign_key	violation of FOREIGN KEY constraint "@1" on table "@2" Нарушение ограничения внешнего ключа @1 для таблицы @2.
-820	335544467	high_minor	minor version too high found @1 expected @2 Не используется.
-901	335544468	tra_state	transaction @1 is @2 Транзакция @1 является @2
-532	335544469	trans_invalid	transaction marked invalid and cannot be committed Транзакция помечена как недействительная из-за ошибки ввода-вывода.
-902	335544470	buf_invalid	cache buffer for page @1 invalid Неверный буфер кэша для страницы @1.
-902	335544471	indexnotdefined	there is no index in table @1 with id @2 Не используется.
-902	335544472	login	Your user name and password are not defined. Ask your database administrator to set up a Firebird login. Не определены ваше имя и пароль пользователя. Чтобы установить соединение с Firebird обратитесь к администратору базы данных.
-823	335544473	invalid_bookmark	invalid bookmark handle Не используется.
-824	335544474	bad_lock_level	invalid lock level @1 Не используется.
-615	335544475	relation_lock	lock on table @1 conflicts with existing lock Блокировка в таблице @1 конфликтует с существующей блокировкой.
-615	335544476	record_lock	requested record lock conflicts with existing lock Запрашиваемая блокировка записи конфликтует с уже существующей блокировкой.
-692	335544477	max_idx	maximum indexes per table (@1) exceeded Превышено максимальное количество индексов для таблицы (@1).
-902	335544478	jrn_enable	enable journal for database before starting online dump Не используется.
-902	335544479	old_failure	online dump failure. Retry dump Не используется.
-902	335544480	old_in_progress	an online dump is already in progress Не используется.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-902	335544481	old_no_space	no more disk/tape space. Cannot continue online dump Не используется.
-902	335544482	no_wal_no_jrn	journaling allowed only if database has Write-ahead Log Не используется.
-902	335544483	num_old_files	maximum number of online dump files that can be specified is 16 Не используется.
-902	335544484	wal_file_open	error in opening Write-ahead Log file during recovery Не используется.
-901	335544485	bad_stmt_handle	invalid statement handle Неверный дескриптор оператора.
-902	335544486	wal_failure	Write-ahead log subsystem failure Не используется.
-230	335544487	walw_err	WAL Writer error Не используется.
-231	335544488	logh_small	Log file header of @1 too small Не используется.
-232	335544489	logh_inv_version	Invalid version of log file @1 Не используется.
-233	335544490	logh_open_flag	Log file @1 not latest in the chain but open flag still set Не используется.
-234	335544491	logh_open_flag2	Log file @1 not closed properly; database recovery may be required Не используется.
-235	335544492	logh_diff_dbname	Database name in the log file @1 is different Не используется.
-236	335544493	logf_unexpected_eof	Unexpected end of log file @1 at offset @2 Не используется.
-237	335544494	logr_incomplete	Incomplete log record at offset @1 in log file @2 Не используется.
-238	335544495	logr_header_small	Log record header too small at offset @1 in log file @2 Не используется.
-239	335544496	logb_small	Log block too small at offset @1 in log file @2 Не используется.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-240	335544497	wal_illegal_attach	Illegal attempt to attach to an uninitialized WAL segment for @1 Не используется.
-241	335544498	wal_invalid_wpb	Invalid WAL parameter block option @1 Не используется.
-242	335544499	wal_err_rollover	Cannot roll over to the next log file @1 Не используется.
-243	335544500	no_wal	database does not use Write-ahead Log База данных не использует запись с упреждением лога.
-615	335544501	drop_wal	cannot drop log file when journaling is enabled Невозможно удалить лог файл, пока включено журналирование.
-204	335544502	stream_not_defined	reference to invalid stream number Ссылка на неверный номер потока.
-244	335544503	wal_subsys_error	WAL subsystem encountered error Не используется.
-245	335544504	wal_subsys_corrupt	WAL subsystem corrupted Не используется.
-902	335544505	no_archive	must specify archive file when enabling long term journal for databases with round-robin log files Не используется.
-902	335544506	shutinprog	database @1 shutdown in progress Выполняется останов базы данных @1
-615	335544507	range_in_use	refresh range number @1 already in use Не используется.
-834	335544508	range_not_found	refresh range number @1 not found Не используется.
-204	335544509	charset_not_found	CHARACTER SET @1 is not defined Набор символов @1 не определен.
-901	335544510	lock_timeout	lock time-out on wait transaction Истечение времени ожидания блокировки для транзакции WAIT.
-204	335544511	prcnotdef	procedure @1 is not defined Процедура @1 не определена.
-170	335544512	prcmismat	Input parameter mismatch for procedure @1 Входные параметры не соответствуют для процедуры @1.
-246	335544513	wal_bugcheck	Database @1: WAL subsystem bug for pid @2@3 Не используется.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-247	335544514	wal_cant_expand	Could not expand the WAL segment for database @1 Не используется.
-204	335544515	codnotdef	status code @1 unknown Неизвестный код состояния @1.
-204	335544516	xcpnotdef	exception @1 not defined Не определено исключение @1.
-836	335544517	except	exception @1 Исключение @1.
-837	335544518	cache_restart	restart shared cache manager Повторный запуск менеджера совместно используемого кэша.
-825	335544519	bad_lock_handle	invalid lock handle Не используется.
-902	335544520	jrn_present	long-term journaling already enabled Не используется.
-248	335544521	wal_err_rollover2	Unable to roll over please see Firebird log. Не используется.
-249	335544522	wal_err_logwrite	WAL I/O error. Please see Firebird log. Не используется.
-250	335544523	wal_err_jrn_comm	WAL writer - Journal server communication error. Please see Firebird log. Не используется.
-251	335544524	wal_err_expansion	WAL buffers cannot be increased. Please see Firebird log. Не используется.
-252	335544525	wal_err_setup	WAL setup error. Please see Firebird log. Не используется.
-253	335544526	wal_err_ww_sync	obsolete Не используется.
-254	335544527	wal_err_ww_start	Cannot start WAL writer for the database @1 Не используется.
-902	335544528	shutdown	database @1 shutdown База данных @1 остановлена.
-553	335544529	existing_priv_mod	cannot modify an existing user privilege Невозможно изменить существующую привилегию пользователя.
-616	335544530	primary_key_ref	Cannot delete PRIMARY KEY being used in FOREIGN KEY definition. Невозможно удалить первичный ключ, который используется в определении внешнего ключа.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-291	335544531	primary_key_notnull	Column used in a PRIMARY constraint must be NOT NULL. Столбец, используемый в ограничении первичного ключа, должен быть NOT NULL.
-204	335544532	ref_cnstrnt_notfound	Name of Referential Constraint not defined in constraints table. Имя ссылочного ограничения не определено в таблице ограничений.
-660	335544533	foreign_key_notfound	Non-existent PRIMARY or UNIQUE KEY specified for FOREIGN KEY. Не существует первичного или уникального ключа, указанного для внешнего ключа.
-292	335544534	ref_cnstrnt_update	Cannot update constraints (RDB$REF_CONSTRAINTS). Нельзя изменять ограничения (RDB$REF_CONSTRAINTS).
-293	335544535	check_cnstrnt_update	Cannot update constraints (RDB$CHECK_CONSTRAINTS). Нельзя изменять ограничения (RDB$REF_CONSTRAINTS).
-294	335544536	check_cnstrnt_del	Cannot delete CHECK constraint entry (RDB$CHECK_CONSTRAINTS) Нельзя удалять запись ограничения CHECK (RDB$REF_CONSTRAINTS).
-618	335544537	integ_index_seg_del	Cannot delete index segment used by an Integrity Constraint Невозможно удалить сегмент индекса, используемого в ограничении целостности.
-618	335544538	integ_index_seg_mod	Cannot update index segment used by an Integrity Constraint Невозможно изменить сегмент индекса, используемого в ограничении целостности.
-616	335544539	integ_index_del	Cannot delete index used by an Integrity Constraint Невозможно удалить индекс, используемый в ограничении целостности.
-616	335544540	integ_index_mod	Cannot modify index used by an Integrity Constraint Невозможно изменить индекс, используемый в ограничении целостности.
-616	335544541	check_trig_del	Cannot delete trigger used by a CHECK Constraint Невозможно удалить триггер, используемый в ограничении CHECK.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-617	335544542	check_trig_update	Cannot update trigger used by a CHECK Constraint Невозможно изменить триггер, используемый в ограничении CHECK.
-616	335544543	cnstrnt_fld_del	Cannot delete column being used in an Integrity Constraint. Невозможно удалить столбец, используемый в ограничении целостности.
-617	335544544	cnstrnt_fld_rename	Cannot rename column being used in an Integrity Constraint. Невозможно переименовать столбец, используемый в ограничении целостности.
-295	335544545	rel_cnstrnt_update	Cannot update constraints (RDB$RELATION_CONSTRAINTS). Нельзя изменять ограничения (RDB$RELATION_CONSTRAINTS).
-150	335544546	constaint_on_view	Cannot define constraints on views Нельзя определить ограничения для представлений.
-296	335544547	invld_cnstrnt_type	internal Firebird consistency check (invalid RDB$CONSTRAINT_TYPE) Внутренняя ошибка программного обеспечения на согласованность (неверный тип RDB$CONSTRAINT_TYPE).
-831	335544548	primary_key_exists	Attempt to define a second PRIMARY KEY for the same table Попытка определения второго первичного ключа для той же таблицы.
-607	335544549	systrig_update	cannot modify or erase a system trigger Невозможно изменить или удалить системный триггер.
-552	335544550	not_rel_owner	only the owner of a table may reassign ownership Не используется.
-204	335544551	grant_obj_notfound	could not find object for GRANT Невозможно найти объект для GRANT.
-205	335544552	grant_fld_notfound	could not find column for GRANT Невозможно найти столбец для GRANT.
-552	335544553	grant_nopriv	user does not have GRANT privileges for operation Пользователь не имеет назначенных привилегий для операции.
-84	335544554	nonsql_security_rel	object has non-SQL security class defined Для объекта определен класс безопасности, не являющийся классом SQL.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-84	335544555	nonsql_security_fld	column has non-SQL security class defined Для столбца определен класс безопасности, не являющийся классом SQL.
-255	335544556	wal_cache_err	Write-ahead Log without shared cache configuration not allowed Не используется.
-902	335544557	shutfail	database shutdown unsuccessful Неуспешный останов базы данных.
-297	335544558	check_constraint	Operation violates CHECK constraint @1 on view or table @2 Операция нарушает ограничение CHECK @1 для представления или таблицы @2.
-901	335544559	bad_svc_handle	invalid service handle Неверный дескриптор сервиса.
-838	335544560	shutwarn	database @1 shutdown in @2 seconds Не используется.
-901	335544561	wrospbver	wrong version of service parameter block Неверная версия блока параметра сервиса.
-901	335544562	bad_spb_form	unrecognized service parameter block Нераспознанный блок параметров сервиса.
-901	335544563	svcnotdef	service @1 is not defined Сервис @1 не определен.
-902	335544564	no_jrn	long-term journaling not enabled Не используется.
-314	335544565	transliteration_failed	Cannot transliterate character between character sets Невозможно выполнить транслитерацию символов между наборами символов.
-257	335544566	start_cm_for_wal	WAL defined; Cache Manager must be started first Не используется.
-258	335544567	wal_ovflow_log_required	Overflow log specification required for round-robin log Не используется.
-204	335544568	text_subtype	Implementation of text subtype @1 not located. Реализация текстового подтипа @1 не обнаружена.
-902	335544569	dsql_error	Dynamic SQL Error Ошибка динамического SQL.
-104	335544570	dsql_command_err	Invalid command Неверная команда.
-103	335544571	dsql_constant_err	Data type for constant unknown Неизвестный тип данных для константы.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-504	335544572	dsql_cursor_err	Invalid cursor reference Недопустимая ссылка на курсор.
-204	335544573	dsql_datatype_err	Data type unknown Неизвестный тип данных.
-502	335544574	dsql_decl_err	Invalid cursor declaration Неверная декларация курсора.
-510	335544575	dsql_cursor_update_err	Cursor @1 is not updatable Курсор @1 является не обновляемым.
-502	335544576	dsql_cursor_open_err	Attempt to reopen an open cursor Попытка переоткрыть открытый курсор.
-501	335544577	dsql_cursor_close_err	Attempt to reclose a closed cursor Попытка перезакрыть закрытый курсор.
-206	335544578	dsql_field_err	Column unknown Столбец неизвестен.
-104	335544579	dsql_internal_err	Internal error Внутренняя ошибка.
-204	335544580	dsql_relation_err	Table unknown Неизвестная таблица.
-204	335544581	dsql_procedure_err	Procedure unknown Неизвестная процедура.
-518	335544582	dsql_request_err	Request unknown Запрос неизвестен.
-804	335544583	dsql_sqlda_err	SQLDA error Ошибка SQLDA
-804	335544584	dsql_var_count_err	Count of read-write columns does not equal count of values Количество столбцов для чтения/записи не равно количеству значений.
-826	335544585	dsql_stmt_handle	Invalid statement handle Не используется.
-804	335544586	dsql_function_err	Function unknown Неизвестная функция.
-206	335544587	dsql_blob_err	Column is not a BLOB Не используется.
-204	335544588	collation_not_found	COLLATION @1 for CHARACTER SET @2 is not defined Тип сортировки @1 для набора символов @2 не определен.
-204	335544589	collation_not_for_-charset	COLLATION @1 is not valid for specified CHARACTER SET Тип сортировки @1 неверный для указанного набора символов.
-104	335544590	dsql_dup_option	Option specified more than once Не используется.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-104	335544591	dsql_tran_err	Unknown transaction option Не используется.
-104	335544592	dsql_invalid_array	Invalid array reference Неверная ссылка на массив.
-604	335544593	dsql_max_arr_dim_- exceeded	Array declared with too many dimensions Объявлен массив со слишком большой размерностью.
-604	335544594	dsql_arr_range_error	Illegal array dimension range Неверный диапазон размерности массива.
-204	335544595	dsql_trigger_err	Trigger unknown Не используется.
-206	335544596	dsql_subselect_err	Subselect illegal in this context Не используется.
-531	335544597	dsql_crdb_prepare_err	Cannot prepare a CREATE DATABASE/SCHEMA statement Невозможно подготовить к выполнению оператор CREATE DATABASE/SCHEMA.
-157	335544598	specify_field_err	must specify column name for view select expression Требуется задать имя столбца для выражения SELECT в представлении.
-158	335544599	num_field_err	number of columns does not match select list Номера столбцов не соответствуют списку выборки SELECT.
-806	335544600	col_name_err	Only simple column names permitted for VIEW WITH CHECK OPTION Только простые имена столбцов допустимы в предложении VIEW WITH CHECK OPTION.
-807	335544601	where_err	No WHERE clause for VIEW WITH CHECK OPTION Нет предложения WHERE для опции VIEW WITH CHECK.
-808	335544602	table_view_err	Only one table allowed for VIEW WITH CHECK OPTION Только одна таблица допустима для использования предложения VIEW WITH CHECK OPTION.
-809	335544603	distinct_err	DISTINCT, GROUP or HAVING not permitted for VIEW WITH CHECK OPTION Не разрешены предложения DISTINCT, GROUP или HAVING в предложении VIEW WITH CHECK OPTION.
-832	335544604	key_field_count_err	FOREIGN KEY column count does not match PRIMARY KEY Количество столбцов внешнего ключа не соответствует первичному ключу.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-810	335544605	subquery_err	No subqueries permitted for VIEW WITH CHECK OPTION Нет подзапросов разрешенных для VIEW WITH CHECK OPTION.
-833	335544606	expression_eval_err	expression evaluation not supported Результат вычисления выражения не поддерживается.
-599	335544607	node_err	gen.c: node not supported Не используется.
-104	335544608	command_end_err	Unexpected end of command Неверное завершение команды.
-901	335544609	index_name	INDEX @1 Индекс @1.
-901	335544610	exception_name	EXCEPTION @1 Исключение @1.
-901	335544611	field_name	COLUMN @1 Столбец @1.
-104	335544612	token_err	Token unknown Неизвестный синтаксический элемент.
-901	335544613	union_err	union not supported Не используется.
-901	335544614	dsql_construct_err	Unsupported DSQL construct Не поддерживаемая конструкция DSQL
-830	335544615	field_aggregate_err	column used with aggregate Не используется.
-829	335544616	field_ref_err	invalid column reference Не используется.
-208	335544617	order_by_err	invalid ORDER BY clause Неверное предложение ORDER BY.
-171	335544618	return_mode_err	Return mode by value not allowed for this data type Вариант возвращаемого значения недоступен для этого типа данных.
-170	335544619	extern_func_err	External functions cannot have more than 10 parameters Внешняя функция не может иметь более 10 параметров.
-204	335544620	alias_conflict_err	alias @1 conflicts with an alias in the same statement Псевдоним @1 конфликтует с псевдонимом в том же операторе.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-204	335544621	procedure_conflict_error	alias @1 conflicts with a procedure in the same statement Алиас @1 конфликтует в процедурой в том же выражении.
-204	335544622	relation_conflict_err	alias @1 conflicts with a table in the same statement Алиас @2 конфликтует с таблицей в том же выражении.
-901	335544623	dsql_domain_err	Illegal use of keyword VALUE Неверное использование ключевого слова VALUE.
-663	335544624	idx_seg_err	segment count of 0 defined for index @1 Количество сегментов равное 0 определено для индекса @1.
-599	335544625	node_name_err	A node name is not permitted in a secondary, shadow, cache or log file name Имя узла не разрешено в имени вторичного фала, файла оперативной копии, в кэше или в имени файла протокола.
-901	335544626	table_name	TABLE @1 Таблица @1.
-901	335544627	proc_name	PROCEDURE @1 Процедура @1
-660	335544628	idx_create_err	cannot create index @1 Невозможно создать индекс @1.
-259	335544629	wal_shadow_err	Write-ahead Log with shadowing configuration not allowed Не используется.
-616	335544630	dependency	there are @1 dependencies Есть @1 зависимостей.
-663	335544631	idx_key_err	too many keys defined for index @1 Слишком много ключей определено для индекса @1.
-597	335544632	dsql_file_length_err	Preceding file did not specify length, so @1 must include starting page number Предыдущий файл не содержит длины, следовательно, предложение @1 должно включать начальный номер страницы.
-598	335544633	dsql_shadow_number_err	Shadow number must be a positive integer Номер оперативной копии должен быть положительным целым числом.
-104	335544634	dsql_token_unk_err	Token unknown - line @1, column @2 Неизвестный синтаксический элемент - строка @1, символ @2.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-204	335544635	dsql_no_relation_alias	there is no alias or table named @1 at this scope level Не существует указанного псевдонима или таблицы с именем @1 на этом уровне видимости.
-204	335544636	indexname	there is no index @1 for table @2 Не существует индекса @1 для таблицы @2.
-281	335544637	no_stream_plan	table @1 is not referenced in plan Таблица @1 не упоминается в плане.
-282	335544638	stream_twice	table @1 is referenced more than once in plan; use aliases to distinguish На таблицу @1 осуществляются ссылки более одного раза в плане; используйте псевдонимы для различения.
-283	335544639	stream_not_found	table @1 is referenced in the plan but not the from list На таблицу @1 есть ссылки в плане, однако она не указана в списке FROM.
-204	335544640	collation_requires_text	Invalid use of CHARACTER SET or COLLATE Неверное использование набора символов или порядка сортировки.
-901	335544641	dsql_domain_not_found	Specified domain or source column @1 does not exist Указанный домен или исходный столбец @1 не существует.
-284	335544642	index_unused	index @1 cannot be used in the specified plan Индекс @1 не может быть использован в указанном плане.
-282	335544643	dsql_self_join	the table @1 is referenced twice; use aliases to differentiate На таблицу @1 осуществляются ссылки дважды; используйте псевдонимы для различения.
-596	335544644	stream_bof	attempt to fetch before the first record in a record stream Попытка сделать выборку, начиная до первой записи в потоке записи.
-595	335544645	stream_crack	the current position is on a crack Не используется.
-601	335544646	db_or_file_exists	database or file exists Не используется.
-401	335544647	invalid_operator	invalid comparison operator for find operation Не используется.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-924	335544648	conn_lost	Connection lost to pipe server Не используется.
-835	335544649	bad_checksum	bad checksum Не используется.
-689	335544650	page_type_err	wrong page type Неверный тип страницы.
-816	335544651	ext_readonly_err	Cannot insert because the file is readonly or is on a read only medium. Невозможно добавление, потому что файл является файлом только для чтения или располагается на устройстве только для чтения.
-811	335544652	sing_select_err	multiple rows in singleton select Множество строк в одиночном операторе SELECT.
-902	335544653	psw_attach	cannot attach to password database Невозможно соединиться с базой данных пароля.
-902	335544654	psw_start_trans	cannot start transaction for password database Невозможно стартовать транзакцию для базы данных пароля.
-827	335544655	invalid_direction	invalid direction for find operation Не используется.
-901	335544656	dsql_var_conflict	variable @1 conflicts with parameter in same procedure Переменная @1 конфликтует с параметром в той же процедуре.
-607	335544657	dsql_no_blob_array	Array/BLOB/DATE data types not allowed in arithmetic Типы данных Массив/BLOB/даты недопустимы для арифметических операций.
-155	335544658	dsql_base_table	@1 is not a valid base table of the specified view Не используется.
-282	335544659	duplicate_base_table	table @1 is referenced twice in view; use an alias to distinguish На таблицу @1 в представлении осуществляются ссылки дважды; используйте псевдонимы для различения.
-282	335544660	view_alias	view @1 has more than one base table; use aliases to distinguish Представление @1 использует более одного раза базовую таблицу; используйте псевдонимы для различения.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-904	335544661	index_root_page_full	cannot add index, index root page is full. Невозможно добавить индекс, корневая страница индекса заполнена.
-204	335544662	dsql_blob_type_unknown	BLOB SUB_TYPE @1 is not defined Подтип BLOB @1 не определен.
-693	335544663	req_max_clones_exceeded	Too many concurrent executions of the same request Слишком много одновременных выполнений одного и того же запроса.
-637	335544664	dsql_duplicate_spec	duplicate specification of @1 - not supported Дублирование спецификации для @1 не поддерживается.
-803	335544665	unique_key_violation	violation of PRIMARY or UNIQUE KEY constraint "@1" on table "@2" Нарушение ограничения "@1" для первичного или уникального ключа для таблицы "@2".
-901	335544666	srvr_version_too_old	server version too old to support all CREATE DATABASE options Не используется.
-909	335544667	drdb_completed_with_errs	drop database completed with errors Удаление базы данных завершилось с ошибками.
-84	335544668	dsql_procedure_use_err	procedure @1 does not return any values Процедура @1 не возвращает никакого значения.
-313	335544669	dsql_count_mismatch	count of column list and variable list do not match Количество столбцов и переменных в списке не соответствует.
-685	335544670	blob_idx_err	attempt to index BLOB column in index @1 Попытка индексации BLOB столбца в индексе @1.
-685	335544671	array_idx_err	attempt to index array column in index @1 Попытка индексации столбца с типом массив в индексе @1.
-663	335544672	key_field_err	too few key columns found for index @1 (incorrect column name?) Слишком много столбцов ключей обнаружено для индекса @1 (неверные имена столбцов?).
-901	335544673	no_delete	cannot delete Удаление невозможно.
-616	335544674	del_last_field	last column in a table cannot be deleted Последний столбец в таблице не может быть удален.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	335544675	sort_err	sort error Ошибка сортировки.
-904	335544676	sort_mem_err	sort error: not enough memory Ошибка сортировки: нет достаточного объема памяти.
-841	335544677	version_err	too many versions Слишком много версий.
-828	335544678	inval_key_posn	invalid key position Неверная позиция ключа.
-690	335544679	no_segments_err	segments not allowed in expression index @1 Сегменты не допускаются в выражении индекса @1.
-600	335544680	crrp_data_err	sort error: corruption in data structure Не используется.
-691	335544681	rec_size_err	new record size of @1 bytes is too big Новый размер записи в @1 байт является слишком большим.
-605	335544682	dsql_field_ref	Inappropriate self-reference of column Недопустимая ссылка столбца на самого себя.
-904	335544683	req_depth_exceeded	request depth exceeded. (Recursive definition?) Превышена глубина запроса (рекурсивное определение?)
-694	335544684	no_field_access	cannot access column @1 in view @2 Не могу получить доступ к столбцу @1 представления @2.
-162	335544685	no_dbkey	dbkey not available for multi-table views Ключ базы данных недоступен для многотабличных представлений.
-839	335544686	jrn_format_err	journal file wrong format Не используется.
-840	335544687	jrn_file_full	intermediate journal file full Не используется.
-519	335544688	dsql_open_cursor_request	The prepare statement identifies a prepare statement with an open cursor Не используется.
-999	335544689	ib_error	Firebird error Не используется.
-260	335544690	cache_redef	Cache redefined Не используется.
-239	335544691	cache_too_small	Insufficient memory to allocate page buffer cache Недостаточно памяти для выделения кэша под буфер страницы.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-260	335544692	log_redef	Log redefined Не используется.
-239	335544693	log_too_small	Log size too small Не используется.
-239	335544694	partition_too_small	Log partition size too small Не используется.
-261	335544695	partition_not_supp	Partitions not supported in series of log file specification Не используется.
-261	335544696	log_length_spec	Total length of a partitioned log must be specified Не используется.
-842	335544697	precision_err	Precision must be from 1 to 18 Точность должна быть в пределах от 1 до 18.
-842	335544698	scale_nogt	Scale must be between zero and precision Масштаб должен быть между нулем и значением точности.
-842	335544699	expec_short	Short integer expected Ожидается короткое целое.
-842	335544700	expec_long	Long integer expected Не используется.
-842	335544701	expec_ushort	Unsigned short integer expected Ожидается беззнаковое короткое целое.
-105	335544702	escape_invalid	Invalid ESCAPE sequence Неверная ESCAPE последовательность.
-901	335544703	svcnoexe	service @1 does not have an associated executable Не используется.
-901	335544704	net_lookup_err	Failed to locate host machine. Не удалось найти хост машины
-901	335544705	service_unknown	Undefined service @1/@2. Не используется.
-901	335544706	host_unknown	The specified name was not found in the hosts file or Domain Name Services. Указанное имя не найдено в файле hosts или в DNS.
-552	335544707	grant_nopriv_on_base	user does not have GRANT privileges on base table/view for operation Пользователь не имеет назначенных привилегий на таблицу/ представление для операции.
-203	335544708	dyn_fld_ambiguous	Ambiguous column reference. Неоднозначная ссылка на столбец.
-104	335544709	dsql_agg_ref_err	Invalid aggregate reference Неверная ссылка на агрегат.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-282	335544710	complex_view	navigational stream @1 references a view with more than one base table Поток навигации @1 ссылается на представление с более чем одной базовой таблицей.
-901	335544711	unprepared_stmt	Attempt to execute an unprepared dynamic SQL statement. Попытка выполнения неподготовленного оператора динамического SQL.
-842	335544712	expec_positive	Positive value expected Ожидается положительное значение.
-804	335544713	dsql_sqlda_value_err	Incorrect values within SQLDA structure Некорректные значения в SQLDA структуре.
-104	335544714	invalid_array_id	invalid blob id Неверный идентификатор BLOB.
-816	335544715	extfile_uns_op	Operation not supported for EXTERNAL FILE table @1 Операция не поддерживается для внешней таблицы @1.
-901	335544716	svc_in_use	Service is currently busy: @1 Сервис в настоящий момент занят: @1
-902	335544717	err_stack_limit	stack size insufficent to execute current request Не используется.
-827	335544718	invalid_key	Invalid key for find operation Неверный ключ для операции поиска.
-901	335544719	net_init_error	Error initializing the network software. Ошибка инициализации сетевого программного обеспечения.
-901	335544720	loadlib_failure	Unable to load required library @1. Не используется.
-902	335544721	network_error	Unable to complete network request to host "@1". Невозможно завершить сетевой запрос на хост "@1"
-902	335544722	net_connect_err	Failed to establish a connection. Ошибка при установлении соединения.
-902	335544723	net_connect_listen_err	Error while listening for an incoming connection. Ошибка при прослушивании входного соединения.
-902	335544724	net_event_connect_err	Failed to establish a secondary connection for event processing. Ошибка при установлении вторичного соединения для обработки события

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-902	335544725	net_event_listen_err	Error while listening for an incoming event connection request. Ошибка при прослушивании запроса события соединения.
-902	335544726	net_read_err	Error reading data from the connection. Ошибка чтения данных из соединения.
-902	335544727	net_write_err	Error writing data to the connection. Ошибка записи данных в соединение
-616	335544728	integ_index_deactivate	Cannot deactivate index used by an integrity constraint Невозможно деактивировать индекс, используемый в ограничении целостности.
-616	335544729	integ_deactivate_primary	Cannot deactivate index used by a PRIMARY/UNIQUE constraint Невозможно деактивировать индекс используемый в ограничении первичного/уникального ключа.
-104	335544730	cse_not_supported	Client/Server Express not supported in this release Не используется.
-901	335544731	tra_must_sweep	- Не используется.
-902	335544732	unsupported_network_-drive	Access to databases on file servers is not supported. Не используется.
-902	335544733	io_create_err	Error while trying to create file Ошибка ввода-вывода при попытке создания файла
-902	335544734	io_open_err	Error while trying to open file Ошибка при попытке открытия файла.
-902	335544735	io_close_err	Error while trying to close file Не используется.
-902	335544736	io_read_err	Error while trying to read from file Ошибка при попытке чтения из файла.
-902	335544737	io_write_err	Error while trying to write to file Ошибка при попытке записи в файл.
-902	335544738	io_delete_err	Error while trying to delete file Ошибка при попытке удаления файла.
-902	335544739	io_access_err	Error while trying to access file Ошибка при попытке доступа к файлу.
-901	335544740	udf_exception	A fatal exception occurred during the execution of a user defined function. Не используется.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	335544741	lost_db_connection	connection lost to database Потеряно соединение с базой данных.
-901	335544742	no_write_user_priv	User cannot write to RDB$USER_PRIVILEGES Пользователь не может писать в таблицу RDB$USER_PRIVILEGES
-104	335544743	token_too_long	token size exceeds limit Размер синтаксического элемента превышает допустимый предел.
-906	335544744	max_att_exceeded	Maximum user count exceeded. Contact your database administrator. Не используется.
-902	335544745	login_same_as_role_name	Your login @1 is same as one of the SQL role name. Ask your database administrator to set up a valid Firebird login. Ваше регистрационное имя @1 совпадает с именем роли SQL. Уточните у вашего администратора базы данных допустимое регистрационное имя Firebird.
-607	335544746	reftable_requires_pk	"REFERENCES table" without "(column)" requires PRIMARY KEY on referenced table "REFERENCES таблица" без указания элемента "(имя столбца)" требует наличие первичного ключа в таблице, на которую осуществляется ссылка.
-85	335544747	usrname_too_long	The username entered is too long. Maximum length is 31 bytes. Введенное имя пользователя очень длинное. Максимальная длина 31 байт.
-85	335544748	password_too_long	The password specified is too long. Maximum length is 8 bytes. Не используется.
-85	335544749	usrname_required	A username is required for this operation. Для этой операции требуется имя пользователя.
-85	335544750	password_required	A password is required for this operation Для этой операции требуется пароль.
-85	335544751	bad_protocol	The network protocol specified is invalid Указан неверный сетевой протокол.
-85	335544752	dup_usrname_found	A duplicate user name was found in the security database Не используется.
-85	335544753	usrname_not_found	The user name specified was not found in the security database Указанное имя пользователя не найдено в базе данных безопасности.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-85	335544754	error_adding_sec_record	An error occurred while attempting to add the user. Обнаружена ошибка при попытке добавления пользователя.
-85	335544755	error_modifying_sec_-record	An error occurred while attempting to modify the user record. Обнаружена ошибка при попытке изменения записи пользователя.
-85	335544756	error_deleting_sec_-record	An error occurred while attempting to delete the user record. Обнаружена ошибка при попытке удаления записи пользователя.
-85	335544757	error_updating_sec_db	An error occurred while updating the security database. Обнаружена ошибка при изменении базы данных безопасности.
-904	335544758	sort_rec_size_err	sort record size of @1 bytes is too big Размер записи сортировки в @1 байт слишком велик
-204	335544759	bad_default_value	can not define a not null column with NULL as default value Нельзя определять непустой столбец (NOT NULL) вместе со значением по умолчанию NULL.
-204	335544760	invalid_clause	invalid clause - '@1' Неверное предложение - '@1'.
-904	335544761	too_many_handles	too many open handles to database Слишком много открытых дескрипторов базы данных.
-904	335544762	optimizer_blk_exc	size of optimizer block exceeded Превышен размер блока оптимизатора.
-104	335544763	invalid_string_constant	a string constant is delimited by double quotes Строковая константа заключена в двойные кавычки.
-104	335544764	transitional_date	DATE must be changed to TIMESTAMP DATE должно быть изменено в TIMESTAMP.
-817	335544765	read_only_database	attempted update on read-only database Попытка изменения базы данных только для чтения.
-817	335544766	must_be_dialect_2_and_-up	SQL dialect @1 is not supported in this database Не используется.
-901	335544767	blob_filter_exception	A fatal exception occurred during the execution of a blob filter. Не используется.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	335544768	exception_access_-violation	Access violation. The code attempted to access a virtual address without privilege to do so. Нарушение доступа. Код пытается получить доступ к виртуальному адресу без соответствующих привилегий на это действие.
-901	335544769	exception_datatype_-missalignment	Datatype misalignment. The attempted to read or write a value that was not stored on a memory boundary. Не используется.
-901	335544770	exception_array_bounds_-exceeded	Array bounds exceeded. The code attempted to access an array element that is out of bounds. Превышены границы размеров массива. Код пытается получить доступ к элементу массива, который находится за пределами его границ.
-901	335544771	exception_float_-denormal_operand	Float denormal operand. One of the floating-point operands is too small to represent a standard float value. Не используется.
-901	335544772	exception_float_divide_-by_zero	Floating-point divide by zero. The code attempted to divide a floating-point value by zero. Деление числа с плавающей точкой на ноль. Код пытается разделить значение с плавающей точкой на ноль.
-901	335544773	exception_float_-inexact_result	Floating-point inexact result. The result of a floating-point operation cannot be represented as a deciaml fraction. Не используется.
-901	335544774	exception_float_-invalid_operand	Floating-point invalid operand. An indeterminant error occurred during a floating-point operation. Не используется.
-901	335544775	exception_float_overflow	Floating-point overflow. The exponent of a floating-point operation is greater than the magnitude allowed. Переполнение числа с плавающей точкой. Экспонента операции с числами с плавающей точкой больше, чем доступные размеры.
-901	335544776	exception_float_stack_-check	Floating-point stack check. The stack overflowed or underflowed as the result of a floating-point operation. Не используется.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	335544777	exception_float_-underflow	Floating-point underflow. The exponent of a floating-point operation is less than the magnitude allowed. Не используется.
-901	335544778	exception_integer_-divide_by_zero	Integer divide by zero. The code attempted to divide an integer value by an integer divisor of zero. Деление целого на ноль. Код пытается разделить целое значение на целый делитель, который является нулем.
-901	335544779	exception_integer_-overflow	Integer overflow. The result of an integer operation caused the most significant bit of the result to carry. Целочисленное переполнение. Результат операции над целыми числами дает больше знаков, чем может храниться в данном результате.
-901	335544780	exception_unknown	An exception occurred that does not have a description. Exception number @1. Не используется.
-901	335544781	exception_stack_overflow	Stack overflow. The resource requirements of the runtime stack have exceeded the memory available to it. Переполнение стека. Требуемые для стека ресурсы во время выполнения исчерпали доступную для этого память.
-901	335544782	exception_sigsegv	Segmentation Fault. The code attempted to access memory without privileges. Не используется.
-901	335544783	exception_sigill	Illegal Instruction. The Code attempted to perfrom an illegal operation. Неверная операция. Код пытается выполнить неверную операцию.
-901	335544784	exception_sigbus	Bus Error. The Code caused a system bus error. Не используется.
-901	335544785	exception_sigfpe	Floating Point Error. The Code caused an Arithmetic Exception or a floating point exception. Не используется.
-901	335544786	ext_file_delete	Cannot delete rows from external files. Невозможно удалять строки из внешних файлов.
-901	335544787	ext_file_modify	Cannot update rows in external files. Невозможно изменять строки во внешних файлах.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	335544788	adm_task_denied	Unable to perform operation. You must be either SYSDBA or owner of the database Невозможно выполнить операцию. Вы должны быть или пользователем SYSDBA, или владельцем базы данных.
-105	335544789	extract_input_mismatch	Specified EXTRACT part does not exist in input datatype Заданный выделяемый элемент в EXTRACT не существует во входном типе данных.
-551	335544790	insufficient_svc_-privileges	Service @1 requires SYSDBA permissions. Reattach to the Service Manager using the SYSDBA account. Сервис @1 требует полномочий SYSDBA. Соединитесь с Менеджером Сервисов как пользователь SYSDBA.
-902	335544791	file_in_use	The file @1 is currently in use by another process. Try again later. Не используется.
-904	335544792	service_att_err	Cannot attach to services manager Невозможно подключиться к менеджеру сервисов.
-817	335544793	ddl_not_allowed_by_db_-sql_dial	Metadata update statement is not allowed by the current database SQL dialect @1 Оператор изменения метаданных недопустим в текущем диалекте SQL базы данных @1.
-901	335544794	cancelled	operation was cancelled Операция была отменена.
-902	335544795	unexp_spb_form	unexpected item in service parameter block, expected @1 Неопределенный элемент в блоке параметров сервиса, ожидается @1.
-104	335544796	sql_dialect_datatype_-unsupport	Client SQL dialect @1 does not support reference to @2 datatype SQL диалект клиента @1 не поддерживает ссылку на тип данных @2.
-901	335544797	svcnouser	user name and password are required while attaching to the services manager Требуются имя пользователя и пароль при подключении к Менеджеру Сервисов.
-104	335544798	depend_on_uncommitted_-rel	You created an indirect dependency on uncommitted metadata. You must roll back the current transaction. Не используется.
-904	335544799	svc_name_missing	The service name was not specified. Не было указано имя сервиса.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-204	335544800	too_many_contexts	Too many Contexts of Relation/Procedure/Views. Maximum allowed is 256 Слишком большой контекст в отношении/процедуре/представлении. Допустимо максимум 256.
-901	335544801	datype_notsup	data type not supported for arithmetic Тип данных не поддерживается для арифметических операций
501	335544802	dialect_reset_warning	Database dialect being changed from 3 to 1 Диалог базы данных изменился с 3 на 1
-901	335544803	dialect_not_changed	Database dialect not changed. Диалект базы данных не был изменен.
-901	335544804	database_create_failed	Unable to create database @1 Невозможно создать базу данных @1.
-901	335544805	inv_dialect_specified	Database dialect @1 is not a valid dialect. Диалект базы данных @1 не является допустимым диалектом.
-901	335544806	valid_db_dialects	Valid database dialects are @1. Допустимыми диалектами базы данных являются @1.
300	335544807	sqlwarn	SQL warning code = @1 Код SQL предупреждения = @1
301	335544808	dtype_renamed	DATE data type is now called TIMESTAMP Тип данных DATE теперь называется TIMESTAMP.
-902	335544809	extern_func_dir_error	Function @1 is in @2, which is not in a permitted directory for external functions. Не используется.
-833	335544810	date_range_exceeded	value exceeds the range for valid dates Значение превышает диапазон допустимых дат.
-901	335544811	inv_client_dialect_-specified	passed client dialect @1 is not a valid dialect. Переданный клиентом диалект @1 не является верным диалектом
-901	335544812	valid_client_dialects	Valid client dialects are @1. Допустимыми клиентскими диалектами являются @1.
-904	335544813	optimizer_between_err	Unsupported field type specified in BETWEEN predicate. Не используется.
-901	335544814	service_not_supported	Services functionality will be supported in a later version of the product Функциональность сервисов будет поддерживаться на более поздних версиях продукта.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-607	335544815	generator_name	GENERATOR @1 Генератор @1.
-607	335544816	udf_name	UDF @1.
-204	335544817	bad_limit_param	Invalid parameter to FETCH or FIRST. Only integers >= 0 are allowed. Неверный параметр для FETCH или FIRST. Допустимы только целые значения >= 0.
-204	335544818	bad_skip_param	Invalid parameter to OFFSET or SKIP. Only integers >= 0 are allowed. Неверный параметр для OFFSET или SKIP. Допустимы только целые значения >= 0.
-902	335544819	io_32bit_exceeded_err	File exceeded maximum size of 2GB. Add another database file or use a 64 bit I/O version of Firebird. Файл превысил максимальный размер 2 Гб. Добавьте другой файл базы данных или используйте 64-битовую версию Firebird.
-901	335544820	invalid_savepoint	Unable to find savepoint with name @1 in transaction context Невозможно найти точку сохранения с именем @1 в контексте транзакции.
-104	335544821	dsql_column_pos_err	Invalid column position used in the @1 clause В предложении @1 используется неверная позиция столбца.
-104	335544822	dsql_agg_where_err	Cannot use an aggregate or window function in a WHERE clause, use HAVING (for aggregate only) instead Невозможно использовать агрегатную или оконную функцию в предложении WHERE. Используйте вместо этого HAVING (только для агрегатных функций).
-104	335544823	dsql_agg_group_err	Cannot use an aggregate or window function in a GROUP BY clause Невозможно использовать агрегатную или оконную функцию в предложении GROUP BY.
-104	335544824	dsql_agg_column_err	Invalid expression in the @1 (not contained in either an aggregate function or the GROUP BY clause) Неверное выражение в @1 (не содержится ни в агрегатной функции, ни в предложении GROUP BY)

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-104	335544825	dsql_agg_having_err	Invalid expression in the @1 (neither an aggregate function nor a part of the GROUP BY clause) Неверное выражение в @1 (не агрегатная функция, и не часть предложения GROUP BY).
-104	335544826	dsql_agg_nested_err	Nested aggregate and window functions are not allowed Вложенные агрегатные или оконные функции недопустимы.
-904	335544827	exec_sql_invalid_arg	Invalid argument in EXECUTE STATEMENT - cannot convert to string Не используется.
-904	335544828	exec_sql_invalid_req	Wrong request type in EXECUTE STATEMENT '@1' Не используется.
-904	335544829	exec_sql_invalid_var	Variable type (position @1) in EXECUTE STATEMENT '@2' INTO does not match returned column type Тип переменной (позиция @1) в EXECUTE STATEMENT '@2' INTO не соответствует возвращаемому типу столбца.
-904	335544830	exec_sql_max_call_-exceeded	Too many recursion levels of EXECUTE STATEMENT Слишком много уровней рекурсии в EXECUTE STATEMENT.
-902	335544831	conf_access_denied	Use of @1 at location @2 is not allowed by server configuration Использование @1 в @2 не разрешено конфигурацией сервера
-904	335544832	wrong_backup_state	Cannot change difference file name while database is in backup mode Невозможно изменить имя дельты пока база данных находится в режиме бэкапа
-904	335544833	wal_backup_err	Physical backup is not allowed while Write-Ahead Log is in use Не используется.
-902	335544834	cursor_not_open	Cursor is not open Курсор не открыт.
-901	335544835	bad_shutdown_mode	Target shutdown mode is invalid for database "@1" Невозможно остановить базу данных "@1".
-802	335544836	concat_overflow	Concatenation overflow. Resulting string cannot exceed 32765 bytes in length Переполнение при конкатенации. Результирующая строка не может превышать 32 Кб.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-204	335544837	bad_substring_offset	Invalid offset parameter @1 to SUBSTRING. Only positive integers are allowed. Неверный параметр смещения @1 для SUBSTRING. Возможны только положительные целые числа.
-530	335544838	foreign_key_target_-doesnt_exist	Foreign key reference target does not exist Ссылки на целевое значение внешнего ключа не существует.
-530	335544839	foreign_key_references_-present	Foreign key references are present for the record Ссылки внешнего ключа присутствуют для записи.
-901	335544840	no_update	cannot update Невозможно обновить.
-902	335544841	cursor_already_open	Cursor is already open Курсор уже открыт.
-901	335544842	stack_trace	@1
-901	335544843	ctx_var_not_found	Context variable @1 is not found in namespace @2 Контекстная переменная @1 не найдена в пространстве имен @2.
-901	335544844	ctx_namespace_invalid	Invalid namespace name @1 passed to @2 Неверное имя пространства имен @1 передается в @2.
-901	335544845	ctx_too_big	Too many context variables Слишком много контекстных переменных.
-901	335544846	ctx_bad_argument	Invalid argument passed to @1 Неверный аргумент передан в @1
-901	335544847	identifier_too_long	BLR syntax error. Identifier @1... is too long Ошибка синтаксиса BLR. Идентификатор @1 является слишком большим
-836	335544848	except2	exception @1 Не используется.
-104	335544849	malformed_string	Malformed string Искаженная (некорректная) строка.
-170	335544850	prc_out_param_mismatch	Output parameter mismatch for procedure @1 Несоответствующие выходные параметры для процедуры @1.
-104	335544851	command_end_err2	Unexpected end of command - line @1, column @2 Неожиданное завершение команды - строка @1, символ @2.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-904	335544852	partner_idx_incompat_-type	partner index segment no @1 has incompatible data type Индексы на внешние ключи имеют несовместимые типы данных в соответствующих сегментах.
-204	335544853	bad_substring_length	Invalid length parameter @1 to SUBSTRING. Negative integers are not allowed. Неверный параметр длины @1 для SUBSTRING. Отрицательные целые числа не доступны.
-204	335544854	charset_not_installed	CHARACTER SET @1 is not installed Набор символов @1 не установлен.
-204	335544855	collation_not_installed	COLLATION @1 for CHARACTER SET @2 is not installed Сортировка @1 для набора символов @2 не установлена.
-902	335544856	att_shutdown	connection shutdown Соединение остановлено.
-904	335544857	blobtoobig	Maximum BLOB size exceeded Достигнут максимальный размер BLOB.
-607	335544858	must_have_phys_field	Can't have relation with only computed fields or constraints Не допустима таблица состоящая только из одних вычисляемых полей или ограничений.
-901	335544859	invalid_time_precision	Time precision exceeds allowed range (0-@1) Уровень точности времени (тип данных TIME) превышает допустимый диапазон (0 - @1)
-413	335544860	blob_convert_error	Unsupported conversion to target type BLOB (subtype @1) Не используется.
-413	335544861	array_convert_error	Unsupported conversion to target type ARRAY Не поддерживается преобразование в массив.
-904	335544862	record_lock_not_supp	Stream does not support record locking Поток не поддерживает блокировку записей.
-904	335544863	partner_idx_not_found	Cannot create foreign key constraint @1. Partner index does not exist or is inactive. Невозможно создать ограничение по внешнему ключу. Индекс-партнер не существует или является неактивным.
-904	335544864	tra_num_exc	Transactions count exceeded. Perform backup and restore to make database operable again Превышено число допустимых транзакций. Выполнете бэкап и рестор, чтобы сделать вновь базу данных работоспособной.
-904	335544865	field_disappeared	Column has been unexpectedly deleted Столбец был неожиданно удален.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	335544866	met_wrong_gtt_scope	@1 cannot depend on @2 @1 не может зависеть от @2.
-204	335544867	subtype_for_internal_use	Blob sub_types bigger than 1 (text) are for internal use only Подтипы BLOB со значением более чем 1 (TEXT) предназначены только для внутреннего использования.
-901	335544868	illegal_prc_type	Procedure @1 is not selectable (it does not contain a SUSPEND statement) Процедура @1 не является процедурой выборки (она не содержит оператор SUSPEND).
-901	335544869	invalid_sort_datatype	Datatype @1 is not supported for sorting operation Тип данных @1 не поддерживает сортировку.
-901	335544870	collation_name	COLLATION @1 Порядок сортировки @1.
-901	335544871	domain_name	DOMAIN @1 Домен @1.
-219	335544872	domnotdef	domain @1 is not defined Домен @1 не определен.
-171	335544873	array_max_dimensions	Array data type can use up to @1 dimensions Тип данных массив не может использовать свыше @1 размерностей.
-901	335544874	max_db_per_trans_allowed	A multi database transaction cannot span more than @1 databases Не используется.
0	335544875	bad_debug_format	Bad debug info format Неверный формат отладочной информации.
-901	335544876	bad_proc_BLR	Error while parsing procedure @1's BLR Ошибка во время разбора BLR процедуры @1.
-901	335544877	key_too_big	index key too big Не используется.
-904	335544878	concurrent_transaction	concurrent transaction number is @1 Число конкурирующих транзакций @1
-625	335544879	not_valid_for_var	validation error for variable @1, value "@2" Ошибка проверки данных для переменной @1, значение "@2".
-625	335544880	not_valid_for	validation error for @1, value "@2" Ошибка проверки данных для @1, значение "@2".
-820	335544881	need_difference	Difference file name should be set explicitly for database on raw device Файл дельты должен быть явно задан для базы данных на raw-устройстве.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-902	335544882	long_login	Login name too long (@1 characters, maximum allowed @2) Слишком длинное имя пользователя (@1 символов, максимально возможно @2)
-205	335544883	fldnotdef2	column @1 is not defined in procedure @2 Столбец @1 не определен в процедуре @2.
-105	335544884	invalid_similar_pattern	Invalid SIMILAR TO pattern Недопустимый шаблон SIMILAR TO.
-901	335544885	bad_teb_form	Invalid TEB format
-901	335544886	tpb_multiple_txn_-isolation	Found more than one transaction isolation in TPB.
-901	335544887	tpb_reserv_before_table	Table reservation lock type @1 requires table name before in TPB.
-901	335544888	tpb_multiple_spec	Found more than one @1 specification in TPB.
-901	335544889	tpb_option_without_rc	Option @1 requires READ COMMITTED isolation in TPB.
-901	335544890	tpb_conflicting_options	Option @1 is not valid if @2 was used previously in TPB.
-901	335544891	tpb_reserv_missing_tlen	Table name length missing after table reservation @1 in TPB.
-901	335544892	tpb_reserv_long_tlen	Table name length @1 is too long after table reservation @2 in TPB.
-901	335544893	tpb_reserv_missing_tname	Table name length @1 without table name after table reservation @2 in TPB.
-901	335544894	tpb_reserv_corrup_tlen	Table name length @1 goes beyond the remaining TPB size after table reservation @2
-901	335544895	tpb_reserv_null_tlen	Table name length is zero after table reservation @1 in TPB.
-901	335544896	tpb_reserv_relnotfound	Table or view @1 not defined in system tables after table reservation @2 in TPB.
-901	335544897	tpb_reserv_-baserelnotfound	Base table or view @1 for view @2 not defined in system tables after table reservation @3 in TPB.
-901	335544898	tpb_missing_len	Option length missing after option @1 in TPB.
-901	335544899	tpb_missing_value	Option length @1 without value after option @2 in TPB.
-901	335544900	tpb_corrupt_len	Option length @1 goes beyond the remaining TPB size after option @2.
-901	335544901	tpb_null_len	Option length is zero after table reservation @1 in TPB.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	335544902	tpb_overflow_len	Option length @1 exceeds the range for option @2 in TPB.
-901	335544903	tpb_invalid_value	Option value @1 is invalid for the option @2 in TPB
-901	335544904	tpb_reserv_stronger_wng	Preserving previous table reservation @1 for table @2, stronger than new @3 in TPB
-901	335544905	tpb_reserv_stronger	Table reservation @1 for table @2 already specified and is stronger than new @3 in TPB
-901	335544906	tpb_reserv_max_recursion	Table reservation reached maximum recursion of @1 when expanding views in TPB
-901	335544907	tpb_reserv_virtualtbl	Table reservation in TPB cannot be applied to @1 because it's a virtual table
-901	335544908	tpb_reserv_systbl	Table reservation in TPB cannot be applied to @1 because it's a system table
-901	335544909	tpb_reserv_temptbl	Table reservation @1 or @2 in TPB cannot be applied to @3 because it's a temporary table
-901	335544910	tpb_readtxn_after_-writelock	Cannot set the transaction in read only mode after a table reservation isc_tpb_lock_write in TPB
-901	335544911	tpb_writelock_after_-readtxn	Cannot take a table reservation isc_tpb_lock_write in TPB because the transaction is in read only mode
-833	335544912	time_range_exceeded	value exceeds the range for a valid time Значение превышает допустимый интервал TIME
-833	335544913	datetime_range_exceeded	value exceeds the range for valid timestamps Значение превышает допустимый интервал TIMESTAMP.
-802	335544914	string_truncation	string right truncation Строка усечена справа.
-802	335544915	blob_truncation	blob truncation when converting to a string: length limit exceeded При конвертации BLOB в строку произошло усечение: превышена допустимая длина.
-802	335544916	numeric_out_of_range	numeric value is out of range Число вышло за пределы диапазона.
-901	335544917	shutdown_timeout	Firebird shutdown is still in progress after the specified timeout Остановка Firebird продолжается по истечении указанного тайм-аута
-901	335544918	att_handle_busy	Attachment handle is busy Не используется.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	335544919	bad_udf_freeit	Bad written UDF detected: pointer returned in FREE_IT function was not allocated by ib_util_malloc Некорректно написана UDF: указатель, возвращаемый функцией FREE_IT, не был выделен функцией ib_util_malloc
-901	335544920	eds_provider_not_found	External Data Source provider '@1' not found Внешний провайдер '@1' не найден.
-901	335544921	eds_connection	Execute statement error at @1 :@2 Data source : @3 Ошибка оператора execute statement в @1 : @2, Data source : @3
-901	335544922	eds_preprocess	Execute statement preprocess SQL error
-901	335544923	eds_stmt_expected	Statement expected
-901	335544924	eds_prm_name_expected	Parameter name expected Ожидается имя параметра.
-901	335544925	eds_unclosed_comment	Unclosed comment found near '@1' Найден незакрытый комментарий около '@1'.
-901	335544926	eds_statement	Execute statement error at @1 :@2Statement : @3Data source : @4
-901	335544927	eds_input_prm_mismatch	Input parameters mismatch Несоответствующие входные параметры.
-901	335544928	eds_output_prm_mismatch	Output parameters mismatch Несоответствующие выходные параметры.
-901	335544929	eds_input_prm_not_set	Input parameter '@1' have no value set Не установлен входной параметр '@1'.
-104	335544930	too_big_blr	BLR stream length @1 exceeds implementation limit @2 Длина потока BLR @1 превышает возможный предел @2.
0	335544931	montabexh	Monitoring table space exhausted Исчерпано пространство таблиц мониторинга.
-172	335544932	modnotfound	module name or entrypoint could not be found Имя модуля или точка входа не найдены.
-901	335544933	nothing_to_cancel	nothing to cancel Нечего отменять.
-901	335544934	ibutil_not_loaded	ib_util library has not been loaded to deallocate memory returned by FREE_IT function Не используется.
-904	335544935	circular_computed	Cannot have circular dependencies with computed fields В вычисляемых полях не может быть циклических зависимостей.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-902	335544936	psw_db_error	Security database error Ошибка в базе данных безопасности.
-833	335544937	invalid_type_datetime_- op	Invalid data type in DATE/TIME/TIMESTAMP addition or subtraction in add_datettime() Неверный тип данных при добавлении или вычитании из DATE/TIME/TIMESTAMP в функции add_datettime().
-833	335544938	onlycan_add_timetodate	Only a TIME value can be added to a DATE value Только значение типа TIME может быть добавлено к DATE.
-833	335544939	onlycan_add_datetotime	Only a DATE value can be added to a TIME value Только значение типа DATE может быть добавлено к TIME.
-833	335544940	onlycansub_- tstampfromtstamp	TIMESTAMP values can be subtracted only from another TIMESTAMP value Значение типа TIMESTAMP можно вычесть только из другого TIMESTAMP значения.
-833	335544941	onlyoneop_mustbe_tstamp	Only one operand can be of type TIMESTAMP Только один операнд может быть типа TIMESTAMP.
-833	335544942	invalid_extractpart_time	Only HOUR, MINUTE, SECOND and MILLISECOND can be extracted from TIME values Только HOUR, MINUTE, SECOND и MILLISECOND могут быть извлечены из значений типа TIME.
-833	335544943	invalid_extractpart_date	HOUR, MINUTE, SECOND and MILLISECOND cannot be extracted from DATE values HOUR, MINUTE, SECOND и MILLISECOND не могут быть извлечены из значений типа DATE.
-833	335544944	invalidarg_extract	Invalid argument for EXTRACT() not being of DATE/TIME/TIMESTAMP type Неверный аргумент для функции EXTRACT(): должен быть один из типов DATE/TIME/TIMESTAMP.
-833	335544945	sysf_argmustbe_exact	Arguments for @1 must be integral types or NUMERIC/DECIMAL without scale Аргументы для @1 должны быть целого типа или NUMERIC/DECIMAL без масштаба.
-833	335544946	sysf_argmustbe_exact_- or_fp	First argument for @1 must be integral type or floating point type Первый аргумент для @1 должен быть целого типа или типа числа с плавающей точкой.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-833	335544947	sysf_argviolates_-uuidtype	Human readable UUID argument for @1 must be of string type Аргумент UUID для @1 должен иметь строковый тип.
-833	335544948	sysf_argviolates_uuidlen	Human readable UUID argument for @2 must be of exact length @1 Аргумент UUID для @2 должен иметь длину @1.
-833	335544949	sysf_argviolates_uuidfmt	Human readable UUID argument for @3 must have "-" at position @2 instead of "@1" Аргумент UUID для @3 должен иметь знак "-" в позиции @2, а не в @1.
-833	335544950	sysf_argviolates_-guidigits	Human readable UUID argument for @3 must have hex digit at position @2 instead of "@1" Аргумент UUID для @3 должен иметь шестнадцатиричное значение в позиции @2 вместо "@1".
-833	335544951	sysf_invalid_addpart_-time	Only HOUR, MINUTE, SECOND and MILLISECOND can be added to TIME values in @1 Только HOUR, MINUTE, SECOND и MILLISECOND могут быть добавлены к значению типа TIME в @1.
-833	335544952	sysf_invalid_add_-datetime	Invalid data type in addition of part to DATE/TIME/TIMESTAMP in @1 Недопустимый тип данных в сложении с DATE/TIME/TIMESTAMP в @1.
-833	335544953	sysf_invalid_addpart_-dtime	Invalid part @1 to be added to a DATE/TIME/TIMESTAMP value in @2 Недопустимое значение @1 добавляется к значению типа DATE/TIME/TIMESTAMP в @2.
-833	335544954	sysf_invalid_add_dtime_-rc	Expected DATE/TIME/TIMESTAMP type in evlDateAdd() result Ожидается тип DATE/TIME/TIMESTAMP на выходе функции evlDateAdd().
-833	335544955	sysf_invalid_diff_dtime	Expected DATE/TIME/TIMESTAMP type as first and second argument to @1 В @1 первый и второй аргумент должны быть типа DATE/TIME/TIMESTAMP.
-833	335544956	sysf_invalid_timediff	The result of TIME-<value> in @1 cannot be expressed in YEAR, MONTH, DAY or WEEK Для совместного использования типа данных TIME с любым другим типом в @1 не разрешается использовать YEAR, MONTH, DAY и WEEK.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-833	335544957	sysf_invalid_-tstamptimediff	The result of TIME-TIMESTAMP or TIMESTAMP-TIME in @1 cannot be expressed in HOUR, MINUTE, SECOND or MILLISECOND. При совместном использовании TIME и TIMESTAMP в @1 результат невозможно выразить в HOUR, MINUTE, SECOND или MILLISECOND.
-833	335544958	sysf_invalid_-datetimediff	The result of DATE-TIME or TIME-DATE in @1 cannot be expressed in HOUR, MINUTE, SECOND and MILLISECOND При совместном использовании TIME и DATE в @1 результат невозможно выразить в HOUR, MINUTE, SECOND и MILLISECOND.
-833	335544959	sysf_invalid_diffpart	Invalid part @1 to express the difference between two DATE/TIME/TIMESTAMP values in @2 Неверная составляющая даты/времени @1, чтобы показать разницу между двумя значениями DATE/TIME/TIMESTAMP @2.
-833	335544960	sysf_argmustbe_positive	Argument for @1 must be positive Аргумент для @1 должен быть положительным.
-833	335544961	sysf_basemustbe_positive	Base for @1 must be positive Основание для @1 должно быть положительным.
-833	335544962	sysf_argnmustbe_nonneg	Argument #@1 for @2 must be zero or positive Аргумент #@1 для @2 должен быть больше или равен нулю.
-833	335544963	sysf_argnmustbe_positive	Argument #@1 for @2 must be positive Аргумент #@1 для @2 должен быть больше нуля.
-833	335544964	sysf_invalid_zeropowneg	Base for @1 cannot be zero if exponent is negative Основание для @1 не может равняться нулю, если показатель степени отрицательный.
-833	335544965	sysf_invalid_negpowfp	Base for @1 cannot be negative if exponent is not an integral value Основание для @1 не может быть отрицательным, когда показатель степени не целое число.
-833	335544966	sysf_invalid_scale	The numeric scale must be between -128 and 127 in @1 Размер числа должен быть между -128 и 127 в @1.
-833	335544967	sysf_argmustbe_nonneg	Argument for @1 must be zero or positive Аргумент для @1 должен быть больше или равен нулю.
-833	335544968	sysf_binuuid_mustbe_str	Binary UUID argument for @1 must be of string type 16-ти байтный UUID для @1 должен быть строкового типа.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-833	335544969	sysf_binuuid_wrongsize	Binary UUID argument for @2 must use @1 bytes 16-ти байтный UUID для @1 должен занимать @1 байт.
-902	335544970	missing_required_spb	Missing required item @1 in service parameter block Отсутствует необходимый элемент @1 в SPB.
-902	335544971	net_server_shutdown	@1 server is shutdown Сервер @1 остановлен.
-924	335544972	bad_conn_str	Invalid connection string Неверная строка подключения.
-901	335544973	bad_epb_form	Unrecognized events block Блок нераспознанных событий.
-902	335544974	no_threads	Could not start first worker thread - shutdown server Не удалось запустить первый рабочий поток - сервер выключен.
-902	335544975	net_event_connect_-timeout	Timeout occurred while waiting for a secondary connection for event processing Произошел таймаут во время ожидания вторичного соединения для обработки событий
-833	335544976	sysf_argmustbe_nonzero	Argument for @1 must be different than zero Аргумент для @1 должен отличаться от нуля.
-833	335544977	sysf_argmustbe_range_-inc1_1	Argument for @1 must be in the range [-1, 1] Аргумент для @1 должен быть в диапазоне [-1, 1]
-833	335544978	sysf_argmustbe_gteq_one	Argument for @1 must be greater or equal than one Аргумент для @1 должен быть больше или равен единице.
-833	335544979	sysf_argmustbe_range_-exc1_1	Argument for @1 must be in the range [-1, 1] Аргумент для @1 должен быть в диапазоне [-1;1].
-104	335544980	internal_rejected_params	Incorrect parameters provided to internal function @1 Неправильные параметры, предоставляемые внутренней функции @1.
-833	335544981	sysf_fp_overflow	Floating point overflow in built-in function @1 Переполнение числа с плавающей точкой во встроенной функции @1.
-901	335544982	udf_fp_overflow	Floating point overflow in result from UDF @1 Переполнение числа с плавающей точкой, возвращаемое UDF @1.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	335544983	udf_fp_nan	Invalid floating point value returned by UDF @1 Неверное число с плавающей точкой, возвращаемое UDF @1.
-902	335544984	instance_conflict	Database is probably already opened by another engine instance in another Windows session База данных, вероятно, уже открыта другим экземпляром сервера в другой сессии Windows
-901	335544985	out_of_temp_space	No free space found in temporary directories Недостаточно свободного места во временных каталогах.
-901	335544986	eds_expl_tran_ctrl	Explicit transaction control is not allowed Явное управление транзакциями запрещено.
-902	335544987	no_trusted_spb	Use of TRUSTED switches in spb_command_line is prohibited Использование переключателя TRUSTED в spb_command_line запрещено.
-901	335544988	package_name	PACKAGE @1 Имя пакета @1.
-901	335544989	cannot_make_not_null	Cannot make field @1 of table @2 NOT NULL because there are NULLs present Невозможно установить полю @1 таблицы @2 ограничение NOT NULL, т.к. в нем присутствуют NULL значения.
-901	335544990	feature_removed	Feature @1 is not supported anymore Это свойство больше не поддерживается.
-901	335544991	view_name	VIEW @1 Имя представления @1.
-904	335544992	lock_dir_access	Can not access lock files directory @1 Не удается получить доступ к каталогу файлов блокировок @1.
-901	335544993	invalid_fetch_option	Fetch option @1 is invalid for a non-scrollable cursor Команда FETCH @1 недоступна для однонаправленных курсоров.
-901	335544994	bad_fun_BLR	Error while parsing function @1's BLR
-901	335544995	func_pack_not_-implemented	Cannot execute function @1 of the unimplemented package @2 Невозможно выполнить функцию @1 нереализованного пакета @2.
-901	335544996	proc_pack_not_-implemented	Cannot execute procedure @1 of the unimplemented package @2 Невозможно выполнить процедуру @1 нереализованного пакета @2.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	335544997	eem_func_not_returned	External function @1 not returned by the external engine plugin @2 Внешняя функция @1 не возвращается плагином @2 для работы с внешними модулями.
-901	335544998	eem_proc_not_returned	External procedure @1 not returned by the external engine plugin @2 Внешняя процедура @1 не возвращается плагином @2 для работы с внешними модулями.
-901	335544999	eem_trig_not_returned	External trigger @1 not returned by the external engine plugin @2 Внешний триггер @1 не возвращается плагином @2 для работы с внешними модулями.
-901	335545000	eem_bad_plugin_ver	Incompatible plugin version @1 for external engine @2 Несовместимая версия плагина @1 для внешнего движка @2
-901	335545001	eem_engine_notfound	External engine @1 not found Движок для работы с внешними модулями @1 не найден.
-532	335545002	attachment_in_use	Attachment is in use Соединение используется.
-532	335545003	transaction_in_use	Transaction is in use Транзакция используется.
-901	335545004	pman_cannot_load_plugin	Error loading plugin @1 Невозможно загрузить плагин @1.
-901	335545005	pman_module_notfound	Loadable module @1 not found Загружаемый модуль @ 1 не найден.
-901	335545006	pman_entrypoint_notfound	Standard plugin entrypoint does not exist in module @1 Стандартная точка входа плагина не существует в модуле @1.
-901	335545007	pman_module_bad	Module @1 exists but can not be loaded Модуль @1 существует, но не может быть загружен.
-901	335545008	pman_plugin_notfound	Module @1 does not contain plugin @2 type @3 Модуль @1 не содержит плагин @2 типа @3.
-833	335545009	sysf_invalid_trig_-namespace	Invalid usage of context namespace DDL_TRIGGER Недопустимое использование пространства имен DDL_TRIGGER.
-901	335545010	unexpected_null	Value is NULL but isNull parameter was not informed Не используется.
-901	335545011	type_notcompat_blob	Type @1 is incompatible with BLOB Не используется.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	335545012	invalid_date_val	Invalid date Не используется.
-901	335545013	invalid_time_val	Invalid time. Не используется.
-901	335545014	invalid_timestamp_val	Invalid timestamp. Не используется.
-901	335545015	invalid_index_val	Invalid index @1 in function @2 Неверный индекс @1 в функции @2.
-836	335545016	formatted_exception	@1
-532	335545017	async_active	Asynchronous call is already running for this attachment Асинхронный вызов уже запущен для этого подключения.
-901	335545018	private_function	Function @1 is private to package @2 Функция @1 частная (private) в пакете @2.
-901	335545019	private_procedure	Procedure @1 is private to package @2 Процедура @1 частная (private) в пакете @2.
-904	335545020	request_outdated	Request can't access new records in relation @1 and should be recompiled Не используется.
-901	335545021	bad_events_handle	invalid events id (handle).
-104	335545022	cannot_copy_stmt	Cannot copy statement @1 Невозможно скопировать оператор @1.
-104	335545023	invalid_boolean_usage	Invalid usage of boolean expression Недопустимое использование булевского выражения.
-833	335545024	sysf_argscant_both_be_-zero	Arguments for @1 cannot both be zero Аргументы для @1 не могут быть равны нулю.
-901	335545025	spb_no_id	missing service ID in spb Отсутствует ID сервиса в SPB.
-901	335545026	ee_blr_mismatch_null	External BLR message mismatch: invalid null descriptor at field @1 Не используется.
-901	335545027	ee_blr_mismatch_length	External BLR message mismatch: length = @1, expected @2 Не используется.
-406	335545028	ss_out_of_bounds	Subscript @1 out of bounds [@2, @3] Индекс @1 вне диапазона [@2, @3].
-902	335545029	missing_data_structures	Install incomplete, please read the Compatibility chapter in the release notes for this version

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-902	335545030	protect_sys_tab	@1 operation is not allowed for system table @2 Операция @1 для системной таблицы @2 не разрешена.
-901	335545031	libtommath_generic	Libtommath error code @1 in function @2 Код ошибки Libtommath @1 в функции @2.
-902	335545032	wroblrver2	unsupported BLR version (expected between @1 and @2, encountered @3) Неподдерживаемая BLR версия (ожидается между @1 и @2, встречена @3).
-551	335545033	trunc_limits	expected length @1, actual @2 Ожидаемая длина @1, фактическая @2
-551	335545034	info_access	Wrong info requested in isc_svc_query() for anonymous service Неверная информация, запрошенная в isc_svc_query() для анонимной службы.
-104	335545035	svc_no_stdin	No isc_info_svc_stdin in user request, but service thread requested stdin data В запросе пользователя нет isc_info_svc_stdin, но поток службы запрашивает данные stdin.
-551	335545036	svc_start_failed	Start request for anonymous service is impossible Запуск анонимного сервиса невозможен.
-104	335545037	svc_no_switches	All services except for getting server log require switches Все сервисы, за исключением запросов логов сервера, требуют переключателей.
-104	335545038	svc_bad_size	Size of stdin data is more than was requested from client Размер данных stdin больше, чем запрашивалось у клиента.
-104	335545039	no_crypt_plugin	Crypt plugin @1 failed to load Плагин шифрования @1 не удалось загрузить
-104	335545040	cp_name_too_long	Length of crypt plugin name should not exceed @1 bytes Длина имени плагина шифрования не должна превышать @1 байт
-901	335545041	cp_process_active	Crypt failed - already crypting database В шифровании отказано - база данных уже шифруется.
-901	335545042	cp_already_crypted	Crypt failed - database is already in requested state В шифровании отказано - база данных уже в требуемом состоянии.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-902	335545043	decrypt_error	Missing crypt plugin, but page appears encrypted Отсутствует плагин шифрования, но страница обозначена как зашифрованная.
-902	335545044	no_providers	No providers loaded Нет загруженных провайдеров
-104	335545045	null_spb	NULL data with non-zero SPB length NULL данные с ненулевой длиной SPB.
-833	335545046	max_args_exceeded	Maximum (@1) number of arguments exceeded for function @2 Превышено максимальное (@1) число аргументов для функции @2.
-901	335545047	ee_blr_mismatch_names_count	External BLR message mismatch: names count = @1, blr count = @2 Не используется.
-901	335545048	ee_blr_mismatch_name_not_found	External BLR message mismatch: name @1 not found Не используется.
-901	335545049	bad_result_set	Invalid resultset interface Недопустимый интерфейс набора данных.
-804	335545050	wrong_message_length	Message length passed from user application does not match set of columns Длина сообщения, переданная из пользовательского приложения, не соответствует набору столбцов.
-804	335545051	no_output_format	Resultset is missing output format information В наборе данных отсутствует информация о выходных данных.
-804	335545052	item_finish	Message metadata not ready - item @1 is not finished Метаданные сообщения не готовы - элемент @1 не завершен.
-902	335545053	miss_config	Missing configuration file: @1 Отсутствует файл конфигурации: @1.
-902	335545054	conf_line	@1: illegal line <@2> @1: недопустимая строка <@2>.
-902	335545055	conf_include	Invalid include operator in @1 for <@2> Недопустимый оператор include в @1 для <@2>.
-902	335545056	include_depth	Include depth too big Глубина include слишком большая.
-902	335545057	include_miss	File to include not found Не найден файл для включения.
-552	335545058	protect_ownership	Only the owner can change the ownership.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	335545059	badvarnum	undefined variable number Неопределенное число переменных
-902	335545060	sec_context	Missing security context for @1 Отсутствует контекст безопасности для @1.
-902	335545061	multi_segment	Missing segment @1 in multisegment connect block parameter Отсутствует сегмент @1 в параметре блока многосегментного соединения.
-902	335545062	login_changed	Different logins in connect and attach packets - client library error Различные логины при подключении и пакетах соединения - ошибка клиентской библиотеки.
-902	335545063	auth_handshake_limit	Exceeded exchange limit during authentication handshake Превышен лимит обмена во время аутентификации
-902	335545064	wirecrypt_incompatible	Incompatible wire encryption levels requested on client and server Несовместимые уровни шифрования сессии, запрашиваемые на клиенте и сервере
-902	335545065	miss_wirecrypt	Client attempted to attach unencrypted but wire encryption is required Клиент пытался подключиться незашифрованно, хотя требуется шифрование сессии.
-902	335545066	wirecrypt_key	Client attempted to start wire encryption using unknown key @1 Клиент попытался запустить шифрование сессии с помощью неизвестного ключа @1.
-902	335545067	wirecrypt_plugin	Client attempted to start wire encryption using unsupported plugin @1 Клиент попытался запустить шифрование сессии с помощью неподдерживаемого плагина @1.
-902	335545068	secdb_name	Error getting security database name from configuration file Ошибка при получении имени базы данных безопасности из файла конфигурации.
-902	335545069	auth_data	Client authentication plugin is missing required data from server Плагин аутентификации на клиенте не содержит требуемых данных с сервера.
-902	335545070	auth_datalength	Client authentication plugin expected @2 bytes of @3 from server, got @1 Плагин аутентификации на клиенте ожидает @2 байтов @3 с сервера, хотя получил @1.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	335545071	info_unprepared_stmt	Attempt to get information about an unprepared dynamic SQL statement. Попытка получить информацию о неподготовленном DSQL операторе.
-901	335545072	idx_key_value	Problematic key value is @1 Сомнительное значение ключа - @1.
-901	335545073	forupdate_virtualtbl	Cannot select virtual table @1 for update WITH LOCK Нельзя использовать FOR UPDATE WITH LOCK для выборки из виртуальной таблицы @1.
-901	335545074	forupdate_systbl	Cannot select system table @1 for update WITH LOCK Нельзя использовать FOR UPDATE WITH LOCK для выборки из системной таблицы @1.
-901	335545075	forupdate_temptbl	Cannot select temporary table @1 for update WITH LOCK Нельзя использовать FOR UPDATE WITH LOCK для выборки из временной таблицы @1.
-901	335545076	cant_modify_sysobj	System @1 @2 cannot be modified Системный объект @1 @2 не может быть изменен.
-901	335545077	server_misconfigured	Server misconfigured - contact administrator please Сервер неверно настроен - свяжитесь с администратором.
-901	335545078	alter_role	Deprecated backward compatibility ALTER ROLE ... SET/DROP AUTO ADMIN mapping may be used only for RDB$ADMIN role Оператор ALTER ROLE ... SET/DROP AUTO ADMIN MAPPING существует только для одной роли - RDB$ADMIN
-901	335545079	map_already_exists	Mapping @1 already exists Отображение @1 уже существует.
-901	335545080	map_not_exists	Mapping @1 does not exist Отображение @1 не существует.
-901	335545081	map_load	@1 failed when loading mapping cache @1 не удалось при загрузке кэша отображения.
-901	335545082	map_aster	Invalid name <*> in authentication block Недопустимое имя <*> в блоке аутентификации.
-901	335545083	map_multi	Multiple maps found for @1 Найдено несколько отображений для @1.
-901	335545084	map_undefined	Undefined mapping result - more than one different results found Неопределенный результат отображения - найдено несколько разных результатов.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-924	335545085	baddpb_damaged_mode	Incompatible mode of attachment to damaged database
-924	335545086	baddpb_buffers_range	Attempt to set in database number of buffers which is out of acceptable range [@1:@2] Попытка установить в базе данных такое количество буферов, которое выходит за допустимый диапазон [@1:@2].
-924	335545087	baddpb_temp_buffers	Attempt to temporarily set number of buffers less than @1 Попытка временно установить количество буферов меньше, чем @1.
-901	335545088	map_nodb	Global mapping is not available when database @1 is not present Глобальное отображение невозможно, если отсутствует база данных @1.
-901	335545089	map_notable	Global mapping is not available when table RDB$MAP is not present in database @1 Глобальное отображение невозможно, если в базе данных @1 отсутствует таблица RDB$MAP.
-901	335545090	miss_trusted_role	Your attachment has no trusted role Ваше подключение не имеет доверительной роли (роли, полученной в результате доверительной аутентификации).
-901	335545091	set_invalid_role	Role @1 is invalid or unavailable Роль @1 недействительна или недоступна.
-596	335545092	cursor_not_positioned	Cursor @1 is not positioned in a valid record
-901	335545093	dup_attribute	Duplicated user attribute @1 Дублированный атрибут пользователя @1.
-901	335545094	dyn_no_priv	There is no privilege for this operation Для этой операции нет привилегий.
-901	335545095	dsql_cant_grant_option	Using GRANT OPTION on @1 not allowed Не разрешено использовать GRANT для @1.
-904	335545096	read_conflict	read conflicts with concurrent update Ошибки чтения при одновременном обновлении.
-901	335545097	crdb_load	@1 failed when working with CREATE DATABASE grants
-901	335545098	crdb_nodb	CREATE DATABASE grants check is not possible when database @1 is not present Проверка на наличие привилегий CREATE DATABASE невозможна, если отсутствует база данных @1.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	335545099	crdb_notable	CREATE DATABASE grants check is not possible when table RDB$DB_CREATORS is not present in database @1 Проверка на наличие привилегий CREATE DATABASE невозможна, если отсутствует таблица RDB$DB_CREATORS в базе данных @1.
-804	335545100	interface_version_too_-old	Interface @3 version too old: expected @1, found @2 Интерфейс @3 слишком устаревший: ожидается @1, найдено @2.
-170	335545101	fun_param_mismatch	Input parameter mismatch for function @1 Несоответствие входных параметров для функции @1.
-901	335545102	savepoint_backout_err	Error during savepoint backout - transaction invalidated Ошибка при возврате в точку сохранения - транзакция недействительна.
-291	335545103	domain_primary_key_-notnull	Domain used in the PRIMARY KEY constraint of table @1 must be NOT NULL Домен, используемый в ограничении PRIMARY KEY таблицы @1, должен быть NOT NULL
-204	335545104	invalid_attachment_-charset	CHARACTER SET @1 cannot be used as a attachment character set Набор символов @1 не может быть использован как набор символов подключения.
-901	335545105	map_down	Some database(s) were shutdown when trying to read mapping data Некоторые базы данных были отключены при попытке считывания данных отображения.
-902	335545106	login_error	Error occurred during login, please check server firebird.log for details Ошибка при входе в систему. Пожалуйста, проверьте firebird.log.
-902	335545107	already_opened	Database already opened with engine instance, incompatible with current База данных уже открыта экземпляром сервера, несовместимым с текущим.
-902	335545108	bad_crypt_key	Invalid crypt key @1 Неверный ключ шифрования @1.
-901	335545109	encrypt_error	Page requires encyption but crypt plugin is missing Страница требует шифрования, но плагин шифрования отсутствует.
-901	335546320	bad_ext_file	Invalid external file format Недопустимый формат внешнего файла.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	335546321	wrong_adp_fields_def	Wrong set of adapter fields Неверный набор полей адаптера
-901	335546322	wrong_adp_field_name	Adapter field named "@1"is not supported Поле адаптера с именем "@1" не поддерживается.
-901	335546323	wrong_adp_field_type	Wrong type of adapter field "@1" Неверный тип поля адаптера "@1".
-901	335546324	bad_ext_record	Unknown record type in external file "@1" at offset @2 Неизвестный тип записи во внешнем файле "@1" при смещении @2.
-901	335546325	bad_adp_type	Unknown adapter type "@1" Неизвестный тип адаптера "@1".
-902	335546326	dir_resolve_error	Cannot resolve directory name "@1" Не удается определить имя каталога "@1".
-902	335546327	read_file_error	Error reading file "@1": @2 Ошибка при чтении файла "@1": @2.
-902	335546328	create_file_error	Error creating file "@1": @2 Ошибка при создании файла "@1": @2.
-902	335546329	delete_file_error	Error deleting file "@1": @2 Ошибка при удалении файла "@1": @2.
-902	335546330	blob_io_error	I/O error during "@1" operation for BLOB file "@2" Ошибка ввода-вывода во время операции "@1" для BLOB файла "@2".
-902	335546331	cant_connect_to_ldap	Cannot connect to LDAP server: @1 Не удается подключиться к LDAP серверу: @1.
-902	335546332	cant_bind_to_ldap	Cannot bind to LDAP server with specified login and password Не удается присоединиться к LDAP серверу с указанным логином и паролем.
-902	335546333	no_ldap_init	libldap or liblber libraries were not loaded correctly Библиотеки libldap или liblber не были загружены корректно.
-833	335546334	sysf_ldap_attr	Cannot get user attribute from LDAP Не удается получить атрибут пользователя из LDAP.
-901	335546335	no_ext_file	Missing external file for adapter Отсутствует внешний файл для адаптера.
-902	335546336	no_gss_init	GSS library was not loaded correctly GSS библиотека была загружена некорректно.
-901	335546337	hash_error	Hash calculation error Ошибка в вычислении хэша.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	335546338	policy_already_exists	Policy @1 already exists Не используется.
-901	335546339	policy_not_exists	Policy @1 does not exist Не используется.
-901	335546340	user_not_exists	User @1 does not exist Не используется.
-901	335546341	policy_is_granted	Policy @1 is granted to a user Не используется.
-901	335546342	bad_trig_BLR	Error while parsing trigger @1's BLR Ошибка при парсинге BLR триггера @1.
-901	335740929	gfix_db_name	data base file name (@1) already given Не используется.
-901	335740930	gfix_invalid_sw	invalid switch @1 Не используется.
-901	335740932	gfix_incmp_sw	incompatible switch combination Не используется.
-901	335740933	gfix_replay_req	replay log pathname required Не используется.
-901	335740934	gfix_pgbuf_req	number of page buffers for cache required Не используется.
-901	335740935	gfix_val_req	numeric value required Не используется.
-901	335740936	gfix_pval_req	positive numeric value required Не используется.
-901	335740937	gfix_trn_req	number of transactions per sweep required Не используется.
-901	335740940	gfix_full_req	"full" or "reserve" required Не используется.
-901	335740941	gfix_usrname_req	user name required Не используется.
-901	335740942	gfix_pass_req	password required Не используется.
-901	335740943	gfix_subs_name	subsystem name Не используется.
-901	335740944	gfix_wal_req	"wal" required Не используется.
-901	335740945	gfix_sec_req	number of seconds required Не используется.
-901	335740946	gfix_nval_req	numeric value between 0 and 32767 inclusive required Не используется.
-901	335740947	gfix_type_shut	must specify type of shutdown Не используется.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	335740948	gfix_retry	please retry, specifying an option Не используется.
-901	335740951	gfix_retry_db	please retry, giving a database name Не используется.
-901	335740991	gfix_exceed_max	internal block exceeds maximum size Не используется.
-901	335740992	gfix_corrupt_pool	corrupt pool Не используется.
-901	335740993	gfix_mem_exhausted	virtual memory exhausted Не используется.
-901	335740994	gfix_bad_pool	bad pool id Не используется.
-901	335740995	gfix_trn_not_valid	Transaction state @1 not in valid range. Не используется.
-901	335741012	gfix_unexp_eoi	unexpected end of input Не используется.
-901	335741018	gfix_recon_fail	failed to reconnect to a transaction in database @1 Не используется.
-901	335741036	gfix_trn_unknown	Transaction description item unknown Не используется.
-901	335741038	gfix_mode_req	"read_only" or "read_write" required Не используется.
-901	335741042	gfix_pzval_req	positive or zero numeric value required Не используется.
-607	336003074	dsql_dbkey_from_non_-table	Cannot SELECT RDB$DB_KEY from a stored procedure. Невозможно выполнить SELECT RDB$DB_KEY из хранимой процедуры.
-104	336003075	dsql_transitional_-numeric	Precision 10 to 18 changed from DOUBLE PRECISION in SQL dialect 1 to 64-bit scaled integer in SQL dialect 3 Не используется.
301	336003076	dsql_dialect_warning_-expr	Use of @1 expression that returns different results in dialect 1 and dialect 3 Выражение @1 возвращает различные значения в диалекте 1 и диалекте 3.
-104	336003077	sql_db_dialect_dtype_-unsupport	Database SQL dialect @1 does not support reference to @2 datatype База данных диалекта SQL @1 не поддерживает ссылку на тип данных @2.
-817	336003079	sql_dialect_conflict_-num	DB dialect @1 and client dialect @2 conflict with respect to numeric precision @3. Не используется.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
301	336003080	dsql_warning_number_-ambiguous	WARNING: Numeric literal @1 is interpreted as a floating-point Предупреждение: числовой литерал @1 интерпретируется как число с плавающей точкой.
301	336003081	dsql_warning_number_-ambiguous1	value in SQL dialect 1, but as an exact numeric value in SQL dialect 3.
301	336003082	dsql_warn_precision_-ambiguous	WARNING: NUMERIC and DECIMAL fields with precision 10 or greater are stored
301	336003083	dsql_warn_precision_-ambiguous1	as approximate floating-point values in SQL dialect 1, but as 64-bit
301	336003084	dsql_warn_precision_-ambiguous2	integers in SQL dialect 3.
-204	336003085	dsql_ambiguous_field_-name	Ambiguous field name between @1 and @2 Двусмысленность имен полей между @1 и @2.
-607	336003086	dsql_udf_return_pos_err	External function should have return position between 1 and @1 Внешняя функция должна иметь позицию возвращаемого значения между 1 и @1.
-104	336003087	dsql_invalid_label	Label @1 @2 in the current scope Метка @1 @2 находится в текущей области видимости.
-104	336003088	dsql_datatypes_not_-comparable	Datatypes @1 are not comparable in expression @2 Типы данных @1 не сравнимы в выражении @2.
-504	336003089	dsql_cursor_invalid	Empty cursor name is not allowed Имя курсора не может быть пустым.
-502	336003090	dsql_cursor_redefined	Statement already has a cursor @1 assigned У операторы уже установлен курсор @1.
-502	336003091	dsql_cursor_not_found	Cursor @1 is not found in the current context Курсор @1 не найден в текущем контексте.
-502	336003092	dsql_cursor_exists	Cursor @1 already exists in the current context Курсор @1 уже существует в текущем контексте.
-502	336003093	dsql_cursor_rel_-ambiguous	Relation @1 is ambiguous in cursor @2 Неоднозначная таблица @1 в курсоре @2.
-502	336003094	dsql_cursor_rel_not_-found	Relation @1 is not found in cursor @2 Таблица @1 не найдена в курсоре @2.
-502	336003095	dsql_cursor_not_open	Cursor is not open Курсор не открыт.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-607	336003096	dsql_type_not_supp_ext_-tab	Data type @1 is not supported for EXTERNAL TABLES. Relation '@2', field '@3' Тип данных @1 не поддерживается для внешних таблиц. Таблица '@2', поле '@3'.
-804	336003097	dsql_feature_not_-supported_ods	Feature not supported on ODS version older than @1.@2 Не используется.
-660	336003098	primary_key_required	Primary key required on table @1 Для таблицы @1 требуется первичный ключ.
-313	336003099	upd_ins_doesnt_match_pk	UPDATE OR INSERT field list does not match primary key of table @1 В UPDATE OR INSERT список полей не соответствует первичному ключу таблицы @1.
-313	336003100	upd_ins_doesnt_match_-matching	UPDATE OR INSERT field list does not match MATCHING clause В UPDATE OR INSERT список полей не соответствует предложению MATCHING.
-817	336003101	upd_ins_with_complex_-view	UPDATE OR INSERT without MATCHING could not be used with views based on more than one table Оператор UPDATE OR INSERT без предложения MATCHING не может быть использован с представлением базирующимися на более чем одной таблице.
-817	336003102	dsql_incompatible_-trigger_type	Incompatible trigger type Несовместимый тип триггера.
-817	336003103	dsql_db_trigger_type_-cant_change	Database trigger type can't be changed Триггер на события базы данных не может быть изменен.
-607	336003104	dsql_record_version_-table	To be used with RDB$RECORD_VERSION, @1 must be a table or a view of single table Для использования с RDB$RECORD_VERSION, @1 должна быть таблицей или представлением одной таблицы
-802	336003105	dsql_invalid_sqlda_-version	SQLDA version expected between @1 and @2, found @3 Версия SQLDA должна быть между @1 и @2, а найдена @3.
-802	336003106	dsql_sqlvar_index	at SQLVAR index @1.
-802	336003107	dsql_no_sqlind	empty pointer to NULL indicator variable Пустой указатель на переменную индикатора NULL
-802	336003108	dsql_no_sqldata	empty pointer to data Пустой указатель на данные.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-802	336003109	dsql_no_input_sqlda	No SQLDA for input values provided Не указана SQLDA для входных значений.
-802	336003110	dsql_no_output_sqlda	No SQLDA for output values provided Не указана SQLDA для выходных значений.
-313	336003111	dsql_wrong_param_num	Wrong number of parameters (expected @1, got @2) Неверное количество параметров (ожидается @1, получено @2).
-817	336004072	dsql_invalid_drop_ss_clause	Invalid DROP SQL SECURITY clause Недействительное условие DROP SQL SECURITY.
-901	336068645	dyn_filter_not_found	BLOB Filter @1 not found Не найден BLOB фильтр @1.
-901	336068649	dyn_func_not_found	Function @1 not found Не найдена функция @1.
-901	336068656	dyn_index_not_found	Index not found Индекс не найден.
-901	336068662	dyn_view_not_found	View @1 not found Представление @1 не найдено.
-901	336068697	dyn_domain_not_found	Domain not found Домен не найден.
-901	336068717	dyn_cant_modify_auto_trig	Triggers created automatically cannot be modified Триггеры, созданные автоматически, не могут быть изменены.
-901	336068740	dyn_dup_table	Table @1 already exists Не используется.
-901	336068748	dyn_proc_not_found	Procedure @1 not found Процедура @1 не найдена.
-901	336068752	dyn_exception_not_found	Exception not found Исключение не найдено.
-901	336068754	dyn_proc_param_not_found	Parameter @1 in procedure @2 not found Параметр @1 в процедуре @2 не найден.
-901	336068755	dyn_trig_not_found	Trigger @1 not found Триггер @1 не найден.
-901	336068759	dyn_charset_not_found	Character set @1 not found Набор символов @1 не найден.
-901	336068760	dyn_collation_not_found	Collation @1 not found Сортировка @1 не найдена.
-901	336068763	dyn_role_not_found	Role @1 not found Роль @1 не найдена.
-901	336068767	dyn_name_longer	Name longer than database column size Имя длиннее, чем размер столбца базы данных.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	336068784	dyn_column_does_not_-exist	column @1 does not exist in table/view @2 Столбца @1 не существует в таблице/представлении @2.
-901	336068796	dyn_role_does_not_exist	SQL role @1 does not exist Не используется.
-901	336068797	dyn_no_grant_admin_opt	user @1 has no grant admin option on SQL role @2 Не используется.
-901	336068798	dyn_user_not_role_member	user @1 is not a member of SQL role @2 Не используется.
-901	336068799	dyn_delete_role_failed	@1 is not the owner of SQL role @2 Не используется.
-901	336068800	dyn_grant_role_to_user	@1 is a SQL role and not a user Не используется.
-901	336068801	dyn_inv_sql_role_name	user name @1 could not be used for SQL role Не используется.
-901	336068802	dyn_dup_sql_role	SQL role @1 already exists Не используется.
-901	336068803	dyn_kywd_spec_for_role	keyword @1 can not be used as a SQL role name Не используется.
-901	336068804	dyn_roles_not_supported	SQL roles are not supported in on older versions of the database. A backup and restore of the database is required. Не используется.
-612	336068812	dyn_domain_name_exists	Cannot rename domain @1 to @2. A domain with that name already exists. Не используется.
-612	336068813	dyn_field_name_exists	Cannot rename column @1 to @2. A column with that name already exists in table @3. Не используется.
-383	336068814	dyn_dependency_exists	Column @1 from table @2 is referenced in @3 Не используется.
-315	336068815	dyn_dtype_invalid	Cannot change datatype for column @1. Changing datatype is not supported for BLOB or ARRAY columns. Не удается изменить тип данных для столбца @1. Изменение типа данных не поддерживается для BLOB полей и полей с массивами.
-829	336068816	dyn_char_fld_too_small	New size specified for column @1 must be at least @2 characters. Новый размер, указанный для столбца @1, должен быть по крайней мере @2 символов.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-829	336068817	dyn_invalid_dtype_-conversion	Cannot change datatype for @1. Conversion from base type @2 to @3 is not supported. Не удается изменить тип данных для @1. Преобразование из базового типа @2 в @3 не поддерживается.
-829	336068818	dyn_dtype_conv_invalid	Cannot change datatype for column @1 from a character type to a non-character type. Невозможно изменить тип данных для столбца @1 из символьного типа в несимвольный.
-901	336068820	dyn_zero_len_id	Zero length identifiers are not allowed Идентификаторы с нулевой длиной не допускаются.
-901	336068822	dyn_gen_not_found	Sequence @1 not found Последовательность @1 не найдена.
-829	336068829	max_coll_per_charset	Maximum number of collations per character set exceeded В наборе символов превышено максимальное число сортировок.
-829	336068830	invalid_coll_attr	Invalid collation attributes Не используется.
-901	336068840	dyn_wrong_gtt_scope	@1 cannot reference @2 Не используется.
-901	336068843	dyn_coll_used_table	Collation @1 is used in table @2 (field name @3) and cannot be dropped Сортировка @1 используется в таблице @2 (поле @3) и не может быть удалена.
-901	336068844	dyn_coll_used_domain	Collation @1 is used in domain @2 and cannot be dropped Сортировка @1 используется в домене @2 и не может быть удалена.
-607	336068845	dyn_cannot_del_syscoll	Cannot delete system collation Невозможно удалить системную сортировку.
-901	336068846	dyn_cannot_del_def_coll	Cannot delete default collation of CHARACTER SET @1 Невозможно удалить сортировку по умолчанию для набора символов @1.
-901	336068849	dyn_table_not_found	Table @1 not found Таблица @1 не найдена.
-901	336068851	dyn_coll_used_procedure	Collation @1 is used in procedure @2 (parameter name @3) and cannot be dropped Сортировка @1 используется в процедуре @2 (имя параметра @3) и не может быть удалена.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-829	336068852	dyn_scale_too_big	New scale specified for column @1 must be at most @2. Новый масштаб, заданный для столбца @1, должен быть не более @2.
-829	336068853	dyn_precision_too_small	New precision specified for column @1 must be at least @2. Новая точность указанная для столбца @1 должна быть по крайней мере @2.
106	336068855	dyn_miss_priv_warning	Warning: @1 on @2 is not granted to @3.
-901	336068856	dyn_ods_not_supp_-feature	Feature '@1' is not supported in ODS @2.@3 Не используется.
-829	336068857	dyn_cannot_addrem_-computed	Cannot add or remove COMPUTED from column @1 Не используется.
-901	336068858	dyn_no_empty_pw	Password should not be empty string Не используется.
-901	336068859	dyn_dup_index	Index @1 already exists Не используется.
-901	336068864	dyn_package_not_found	Package @1 not found Пакет @1 не найден.
-901	336068865	dyn_schema_not_found	Schema @1 not found Схема @1 не найдена.
-607	336068866	dyn_cannot_mod_sysproc	Cannot ALTER or DROP system procedure @1 Невозможно изменить или удалить системную процедуру @1.
-607	336068867	dyn_cannot_mod_systrig	Cannot ALTER or DROP system trigger @1 Невозможно изменить или удалить системный триггер @1.
-607	336068868	dyn_cannot_mod_sysfunc	Cannot ALTER or DROP system function @1 Невозможно изменить или удалить системную функцию @1.
-607	336068869	dyn_invalid_ddl_proc	Invalid DDL statement for procedure @1 Недопустимый оператор DDL для процедуры @1.
-607	336068870	dyn_invalid_ddl_trig	Invalid DDL statement for trigger @1 Недопустимый оператор DDL для триггера @1.
-901	336068871	dyn_funcnotdef_package	Function @1 has not been defined on the package body @2 Функция @1 не определена в теле пакета @2.
-901	336068872	dyn_procnotdef_package	Procedure @1 has not been defined on the package body @2 Процедура @1 не определена в теле пакета @2.
-901	336068873	dyn_funcsignat_package	Function @1 has a signature mismatch on package body @2 Сигнатура функции @1 не соответствует сигнатуре в теле пакета @2.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	336068874	dyn_procsignat_package	Procedure @1 has a signature mismatch on package body @2 Сигнатура процедуры @1 не соответствует сигнатуре в теле пакета @2.
-901	336068875	dyn_defvaldecl_package_proc	Default values for parameters are allowed only in declaration of packaged procedure @1.@2 Значения по умолчанию для параметров разрешены только при объявлении процедуры в пакете @1.@2
-901	336068877	dyn_package_body_exists	Package body @1 already exists Тело пакета @1 уже существует.
-607	336068878	dyn_invalid_ddl_func	Invalid DDL statement for function @1 Недопустимый оператор DDL для функции @1.
-901	336068879	dyn_newfc_oldsyntax	Cannot alter new style function @1 with ALTER EXTERNAL FUNCTION. Use ALTER FUNCTION instead. Невозможно изменить функцию @1 с помощью оператора ALTER EXTERNAL FUNCTION. Вместо этого используйте ALTER FUNCTION.
-901	336068886	dyn_func_param_not_found	Parameter @1 in function @2 not found Параметр @1 в функции @2 не найден.
-901	336068887	dyn_routine_param_not_found	Parameter @1 of routine @2 not found Параметр @1 процедуры/функции @2 не найден.
-901	336068888	dyn_routine_param_ambiguous	Parameter @1 of routine @2 is ambiguous (found in both procedures and functions). Use a specifier keyword. Параметр @1 процедуры/функции @2 неоднозначен (найден как в процедурах, так и в функциях).
-901	336068889	dyn_coll_used_function	Collation @1 is used in function @2 (parameter name @3) and cannot be dropped Сортировка @1 используется в функции @2 (параметр @3) и не может быть удалена.
-901	336068890	dyn_domain_used_function	Domain @1 is used in function @2 (parameter name @3) and cannot be dropped Домен @1 используется в функции @2 (параметр @3) и не может быть удален.
-901	336068891	dyn_alter_user_no_clause	ALTER USER requires at least one clause to be specified Не используется.
-901	336068894	dyn_duplicate_package_item	Duplicate @1 @2 Дублирование @1 @2.
-901	336068895	dyn_cant_modify_sysobj	System @1 @2 cannot be modified Системный объект @1 @2 не может быть изменен.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	336068896	dyn_cant_use_zero_-increment	INCREMENT BY 0 is an illegal option for sequence @1 Шаг приращения в предложении INCREMENT BY не может быть равен нулю для последовательности @1.
-901	336068897	dyn_cant_use_in_-foreignkey	Can't use @1 in FOREIGN KEY constraint Не используется.
-901	336068898	dyn_defvaldecl_package_-func	Default values for parameters are allowed only in declaration of packaged function @1.@2 Значения по умолчанию для параметров разрешены только при объявлении функции в пакете @1.@2
-901	336330753	gbak_unknown_switch	found unknown switch Не используется.
-901	336330754	gbak_page_size_missing	page size parameter missing Не используется.
-901	336330755	gbak_page_size_toobig	Page size specified (@1) greater than limit (16384 bytes) Не используется.
-901	336330756	gbak_redir_ouput_missing	redirect location for output is not specified Не используется.
-901	336330757	gbak_switches_conflict	conflicting switches for backup/restore Не используется.
-901	336330758	gbak_unknown_device	device type @1 not known Не используется.
-901	336330759	gbak_no_protection	protection is not there yet Не используется.
-901	336330760	gbak_page_size_not_-allowed	page size is allowed only on restore or create Не используется.
-901	336330761	gbak_multi_source_dest	multiple sources or destinations specified Не используется.
-901	336330762	gbak_filename_missing	requires both input and output filenames Не используется.
-901	336330763	gbak_dup_inout_names	input and output have the same name. Disallowed. Не используется.
-901	336330764	gbak_inv_page_size	expected page size, encountered "@1" Не используется.
-901	336330765	gbak_db_specified	REPLACE specified, but the first file @1 is a database Не используется.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	336330766	gbak_db_exists	database @1 already exists. To replace it, use the -REP switch Не используется.
-901	336330767	gbak_unk_device	device type not specified Не используется.
-901	336330772	gbak_blob_info_failed	gds_$blob_info failed Не используется.
-901	336330773	gbak_unk_blob_item	do not understand BLOB INFO item @1 Не используется.
-901	336330774	gbak_get_seg_failed	gds_$get_segment failed Не используется.
-901	336330775	gbak_close_blob_failed	gds_$close_blob failed Не используется.
-901	336330776	gbak_open_blob_failed	gds_$open_blob failed Не используется.
-901	336330777	gbak_put_blr_gen_id_-failed	Failed in put_blr_gen_id Ошибка в put_blr_gen_id.
-901	336330778	gbak_unk_type	data type @1 not understood Не используется.
-901	336330779	gbak_comp_req_failed	gds_$compile_request failed Не используется.
-901	336330780	gbak_start_req_failed	gds_$start_request failed Не используется.
-901	336330781	gbak_rec_failed	gds_$receive failed Не используется.
-901	336330782	gbak_rel_req_failed	gds_$release_request failed Не используется.
-901	336330783	gbak_db_info_failed	gds_$database_info failed Не используется.
-901	336330784	gbak_no_db_desc	Expected database description record Не используется.
-901	336330785	gbak_db_create_failed	failed to create database @1 Не используется.
-901	336330786	gbak_decomp_len_error	RESTORE: decompression length error Не используется.
-901	336330787	gbak_tbl_missing	cannot find table @1 Не используется.
-901	336330788	gbak_blob_col_missing	Cannot find column for BLOB Не используется.
-901	336330789	gbak_create_blob_failed	gds_$create_blob failed Не используется.
-901	336330790	gbak_put_seg_failed	gds_$put_segment failed Не используется.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	336330791	gbak_rec_len_exp	expected record length Не используется.
-901	336330792	gbak_inv_rec_len	wrong length record, expected @1 encountered @2 Не используется.
-901	336330793	gbak_exp_data_type	expected data attribute Не используется.
-901	336330794	gbak_gen_id_failed	Failed in store_blr_gen_id Не используется.
-901	336330795	gbak_unk_rec_type	do not recognize record type @1 Не используется.
-901	336330796	gbak_inv_bkup_ver	Expected backup version 1..10. Found @1 Не используется.
-901	336330797	gbak_missing_bkup_desc	expected backup description record Не используется.
-901	336330798	gbak_string_trunc	string truncated Не используется.
-901	336330799	gbak_cant_rest_record	warning - record could not be restored Не используется.
-901	336330800	gbak_send_failed	gds_$send failed Не используется.
-901	336330801	gbak_no_tbl_name	no table name for data Не используется.
-901	336330802	gbak_unexp_eof	unexpected end of file on backup file Не используется.
-901	336330803	gbak_db_format_too_old	database format @1 is too old to restore to Не используется.
-901	336330804	gbak_inv_array_dim	array dimension for column @1 is invalid Не используется.
-901	336330807	gbak_xdr_len_expected	Expected XDR record length Не используется.
-901	336330817	gbak_open_bkup_error	cannot open backup file @1 Не используется.
-901	336330818	gbak_open_error	cannot open status and error output file @1 Не используется.
-901	336330934	gbak_missing_block_fac	blocking factor parameter missing Не используется.
-901	336330935	gbak_inv_block_fac	expected blocking factor, encountered "@1" Не используется.
-901	336330936	gbak_block_fac_specified	a blocking factor may not be used in conjunction with device CT Не используется.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	336330940	gbak_missing_username	user name parameter missing Не используется.
-901	336330941	gbak_missing_password	password parameter missing Не используется.
-901	336330952	gbak_missing_skipped_-bytes	missing parameter for the number of bytes to be skipped Не используется.
-901	336330953	gbak_inv_skipped_bytes	expected number of bytes to be skipped, encountered "@1" Не используется.
-901	336330965	gbak_err_restore_charset	character set Не используется.
-901	336330967	gbak_err_restore_-collation	collation Не используется.
-901	336330972	gbak_read_error	Unexpected I/O error while reading from backup file Не используется.
-901	336330973	gbak_write_error	Unexpected I/O error while writing to backup file Не используется.
-901	336330985	gbak_db_in_use	could not drop database @1 (database might be in use) Не используется.
-901	336330990	gbak_sysmemex	System memory exhausted Не используется.
-901	336331002	gbak_restore_role_failed	SQL role Не используется.
-901	336331005	gbak_role_op_missing	SQL role parameter missing Не используется.
-901	336331010	gbak_page_buffers_-missing	page buffers parameter missing Не используется.
-901	336331011	gbak_page_buffers_-wrong_param	expected page buffers, encountered "@1" Не используется.
-901	336331012	gbak_page_buffers_-restore	page buffers is allowed only on restore or create Не используется.
-901	336331014	gbak_inv_size	size specification either missing or incorrect for file @1 Не используется.
-901	336331015	gbak_file_outof_sequence	file @1 out of sequence Не используется.
-901	336331016	gbak_join_file_missing	can't join - one of the files missing Не используется.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	336331017	gbak_stdin_not_supptd	standard input is not supported when using join operation Не используется.
-901	336331018	gbak_stdout_not_supptd	standard output is not supported when using split operation or in verbose mode Не используется.
-901	336331019	gbak_bkup_corrupt	backup file @1 might be corrupt Не используется.
-901	336331020	gbak_unk_db_file_spec	database file specification missing Не используется.
-901	336331021	gbak_hdr_write_failed	can't write a header record to file @1 Не используется.
-901	336331022	gbak_disk_space_ex	free disk space exhausted Не используется.
-901	336331023	gbak_size_lt_min	file size given (@1) is less than minimum allowed (@2) Не используется.
-901	336331025	gbak_svc_name_missing	service name parameter missing Не используется.
-901	336331026	gbak_not_ownr	Cannot restore over current database, must be SYSDBA or owner of the existing database. Не используется.
-901	336331031	gbak_mode_req	"read_only" or "read_write" required Не используется.
-901	336331033	gbak_just_data	just data ignore all constraints etc. Не используется.
-901	336331034	gbak_data_only	restoring data only ignoring foreign key, unique, not null other constraints Не используется.
-901	336331078	gbak_missing_interval	verbose interval value parameter missing Не используется.
-901	336331079	gbak_wrong_interval	verbose interval value cannot be smaller than @1 Не используется.
-901	336331081	gbak_verify_verbint	verify (verbose) and verbint options are mutually exclusive Не используется.
-901	336331082	gbak_option_only_-restore	option -@1 is allowed only on restore or create Не используется.
-901	336331083	gbak_option_only_backup	option -@1 is allowed only on backup Не используется.
-901	336331084	gbak_option_conflict	options -@1 and -@2 are mutually exclusive Не используется.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	336331085	gbak_param_conflict	parameter for option -@1 was already specified with value "@2" Не используется.
-901	336331086	gbak_option_repeated	option -@1 was already specified Не используется.
-901	336331091	gbak_max_dbkey_-recursion	dependency depth greater than @1 for view @2 Не используется.
-901	336331092	gbak_max_dbkey_length	value greater than @1 when calculating length of rdb$db_key for view @2 Не используется.
-901	336331093	gbak_invalid_metadata	Invalid metadata detected. Use -FIX_FSS_METADATA option. Обнаружены недопустимые метаданные. Используйте параметр -FIX_FSS_METADATA.
-901	336331094	gbak_invalid_data	Invalid data detected. Use -FIX_FSS_DATA option. Обнаружены недопустимые данные. Используйте параметр -FIX_FSS_DATA.
-901	336331096	gbak_inv_bkup_ver2	Expected backup version @2..@3. Found @1 Не используется.
-901	336331100	gbak_db_format_too_old2	database format @1 is too old to backup Не используется.
-804	336397205	dsql_too_old_ods	ODS versions before ODS@1 are not supported Не используется.
-607	336397206	dsql_table_not_found	Table @1 does not exist Таблица @1 не существует.
-607	336397207	dsql_view_not_found	View @1 does not exist Представление @1 не существует.
-206	336397208	dsql_line_col_error	At line @1, column @2 Ошибка в строке @1, столбец @2.
-206	336397209	dsql_unknown_pos	At unknown line and column В неизвестных строке и столбце.
-206	336397210	dsql_no_dup_name	Column @1 cannot be repeated in @2 statement Столбец @1 не может быть повторен в операторе @2.
-901	336397211	dsql_too_many_values	Too many values (more than @1) in member list to match against
-607	336397212	dsql_no_array_computed	Array and BLOB data types not allowed in computed field Не используется.
-637	336397213	dsql_implicit_domain_-name	Implicit domain name @1 not allowed in user created domain Неявное доменное имя @1 не разрешено для доменов, создаваемых пользователями.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-607	336397214	dsql_only_can_-subscript_array	scalar operator used on field @1 which is not an array Скалярный оператор, используемый в поле @1, которое не является массивом.
-104	336397215	dsql_max_sort_items	cannot sort on more than 255 items Не удается сортировать более, чем 255 элементов.
-104	336397216	dsql_max_group_items	cannot group on more than 255 items Не могу группировать более, чем 255 элементов.
-104	336397217	dsql_conflicting_sort_-field	Cannot include the same field (@1.@2) twice in the ORDER BY clause with conflicting sorting options Не используется.
-104	336397218	dsql_derived_table_-more_columns	column list from derived table @1 has more columns than the number of items in its SELECT statement Список столбцов в производной таблице @1 содержит больше столбцов, чем количество элементов в SELECT.
-104	336397219	dsql_derived_table_-less_columns	column list from derived table @1 has less columns than the number of items in its SELECT statement Список столбцов в производной таблице @1 имеет меньше столбцов, чем количество элементов в SELECT.
-104	336397220	dsql_derived_field_-unnamed	no column name specified for column number @1 in derived table @2 Нет имени столбца указанного для столбца под номером @1 в производной таблице @2.
-104	336397221	dsql_derived_field_dup_-name	column @1 was specified multiple times for derived table @2 Столбец @1 был указан несколько раз для производной таблицы @2.
-104	336397222	dsql_derived_alias_-select	Internal dsql error: alias type expected by pass1_expand_select_node Внутренняя ошибка DSQL: тип алиаса ожидался pass1_expand_select_node.
-104	336397223	dsql_derived_alias_-field	Internal dsql error: alias type expected by pass1_field Внутренняя ошибка DSQL: тип алиаса ожидался pass1_field.
-104	336397224	dsql_auto_field_bad_pos	Internal dsql error: column position out of range in pass1_union_auto_cast Внутренняя ошибка DSQL: позиция столбца вышла из диапазона в pass1_union_auto_cast.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-104	336397225	dsql_cte_wrong_reference	Recursive CTE member (@1) can refer itself only in FROM clause Рекурсивная часть CTE (@1) может ссылаться сама на себя только в предложении FROM.
-104	336397226	dsql_cte_cycle	CTE '@1' has cyclic dependencies CTE '@1' имеет циклические зависимости.
-104	336397227	dsql_cte_outer_join	Recursive member of CTE can't be member of an outer join Рекурсивная часть CTE не может являться частью внешнего соединения.
-104	336397228	dsql_cte_mult_-references	Recursive member of CTE can't reference itself more than once Рекурсивная часть CTE не может ссылаться на себя более одного раза.
-104	336397229	dsql_cte_not_a_union	Recursive CTE (@1) must be an UNION Рекурсивный CTE (@1) должен содержать UNION.
-104	336397230	dsql_cte_nonrecurs_-after_recurs	CTE '@1' defined non-recursive member after recursive В CTE '@1' определена не рекурсивная часть после рекурсии.
-104	336397231	dsql_cte_wrong_clause	Recursive member of CTE '@1' has @2 clause Рекурсивная часть CTE '@1' содержит предложение @2.
-104	336397232	dsql_cte_union_all	Recursive members of CTE (@1) must be linked with another members via UNION ALL Рекурсивная часть CTE (@1) должна быть связана с остальными частями через UNION ALL.
-104	336397233	dsql_cte_miss_-nonrecursive	Non-recursive member is missing in CTE '@1' Не рекурсивная часть отсутствует в CTE '@1'.
-104	336397234	dsql_cte_nested_with	WITH clause can't be nested Предложение WITH не может быть вложенным.
-104	336397235	dsql_col_more_than_-once_using	column @1 appears more than once in USING clause Столбец @1 появляется более одного раза в разделе USING.
-901	336397236	dsql_unsupp_feature_-dialect	feature is not supported in dialect @1 Это свойство не поддерживается в диалекте @1
-104	336397237	dsql_cte_not_used	CTE "@1" is not used in query CTE "@1" не используется в запросе.
-104	336397238	dsql_col_more_than_-once_view	column @1 appears more than once in ALTER VIEW Столбец @1 появляется более одного раза в ALTER VIEW.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	336397239	dsql_unsupported_in_-auto_trans	@1 is not supported inside IN AUTONOMOUS TRANSACTION block @1 не поддерживается внутри блока IN AUTONOMOUS TRANSACTION.
-833	336397240	dsql_eval_unknode	Unknown node type @1 in dsql/GEN_expr Не используется.
-833	336397241	dsql_agg_wrongarg	Argument for @1 in dialect 1 must be string or numeric Аргумент для @1 в диалекте 1 должен быть строковым или числовым.
-833	336397242	dsql_agg2_wrongarg	Argument for @1 in dialect 3 must be numeric Аргумент для @1 в диалекте 3 должен быть числовым.
-833	336397243	dsql_nodateortime_pm_-string	Strings cannot be added to or subtracted from DATE or TIME types Строки не могут быть добавлены или вычтены из DATE или TIME.
-833	336397244	dsql_invalid_datetime_-subtract	Invalid data type for subtraction involving DATE, TIME or TIMESTAMP types Недопустимый тип данных для вычитания с использованием типов DATE, TIME или TIMESTAMP
-833	336397245	dsql_invalid_-dateortime_add	Adding two DATE values or two TIME values is not allowed Сложение двух значений DATE или двух значений TIME не допускается.
-833	336397246	dsql_invalid_type_-minus_date	DATE value cannot be subtracted from the provided data type Значение DATE не может быть вычтено из предоставленного типа данных.
-833	336397247	dsql_nostring_addsub_-dial3	Strings cannot be added or subtracted in dialect 3 Строки нельзя складывать или вычитать в диалекте 3.
-833	336397248	dsql_invalid_type_-addsub_dial3	Invalid data type for addition or subtraction in dialect 3 Недопустимый тип данных для сложения или вычитания в диалекте 3.
-833	336397249	dsql_invalid_type_-multip_dial1	Invalid data type for multiplication in dialect 1 Недопустимый тип данных для умножения в диалекте 1.
-833	336397250	dsql_nostring_multip_-dial3	Strings cannot be multiplied in dialect 3 В диалекте 3 строки умножать нельзя.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-833	336397251	dsql_invalid_type_-multip_dial3	Invalid data type for multiplication in dialect 3 Недопустимый тип данных для умножения в диалекте 3.
-833	336397252	dsql_mustuse_numeric_-div_dial1	Division in dialect 1 must be between numeric data types Деление в диалекте 3 должно быть между числовыми типа данных.
-833	336397253	dsql_nostring_div_dial3	Strings cannot be divided in dialect 3 В диалекте 3 сроки делить нельзя.
-833	336397254	dsql_invalid_type_div_-dial3	Invalid data type for division in dialect 3 Недопустимый тип данных для деления в диалекте 3.
-833	336397255	dsql_nostring_neg_dial3	Strings cannot be negated (applied the minus operator) in dialect 3 Нельзя ставить знак «минус» перед строками в диалекте 3
-833	336397256	dsql_invalid_type_neg	Invalid data type for negation (minus operator) Невозможно поставить знак «минус» перед указанным типом данных.
-104	336397257	dsql_max_distinct_items	Cannot have more than 255 items in DISTINCT list Невозможно содержать более 255 элементов в списке DISTINCT.
-901	336397258	dsql_alter_charset_-failed	ALTER CHARACTER SET @1 failed Ошибка в операторе ALTER CHARACTER SET @1.
-901	336397259	dsql_comment_on_failed	COMMENT ON @1 failed Ошибка в операторе COMMENT ON @1.
-901	336397260	dsql_create_func_failed	CREATE FUNCTION @1 failed Ошибка в операторе CREATE FUNCTION @1.
-901	336397261	dsql_alter_func_failed	ALTER FUNCTION @1 failed Ошибка в операторе ALTER FUNCTION @1.
-901	336397262	dsql_create_alter_func_-failed	CREATE OR ALTER FUNCTION @1 failed Ошибка в операторе CREATE OR ALTER FUNCTION @1.
-901	336397263	dsql_drop_func_failed	DROP FUNCTION @1 failed Ошибка в операторе DROP FUNCTION @1.
-901	336397264	dsql_recreate_func_-failed	RECREATE FUNCTION @1 failed Ошибка в операторе RECREATE FUNCTION @1.
-901	336397265	dsql_create_proc_failed	CREATE PROCEDURE @1 failed Ошибка в операторе CREATE PROCEDURE @1.
-901	336397266	dsql_alter_proc_failed	ALTER PROCEDURE @1 failed Ошибка в операторе ALTER PROCEDURE @1.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	336397267	dsql_create_alter_proc_-failed	CREATE OR ALTER PROCEDURE @1 failed Ошибка в операторе CREATE OR ALTER PROCEDURE @1.
-901	336397268	dsql_drop_proc_failed	DROP PROCEDURE @1 failed Ошибка в операторе DROP PROCEDURE @1.
-901	336397269	dsql_recreate_proc_-failed	RECREATE PROCEDURE @1 failed Ошибка в операторе RECREATE PROCEDURE @1.
-901	336397270	dsql_create_trigger_-failed	CREATE TRIGGER @1 failed Ошибка в операторе CREATE TRIGGER @1.
-901	336397271	dsql_alter_trigger_-failed	ALTER TRIGGER @1 failed Ошибка в операторе ALTER TRIGGER @1.
-901	336397272	dsql_create_alter_-trigger_failed	CREATE OR ALTER TRIGGER @1 failed Ошибка в операторе CREATE OR ALTER TRIGGER @1.
-901	336397273	dsql_drop_trigger_-failed	DROP TRIGGER @1 failed Ошибка в операторе DROP TRIGGER @1.
-901	336397274	dsql_recreate_trigger_-failed	RECREATE TRIGGER @1 failed Ошибка в операторе RECREATE TRIGGER @1.
-901	336397275	dsql_create_collation_-failed	CREATE COLLATION @1 failed Ошибка в операторе CREATE COLLATION @1.
-901	336397276	dsql_drop_collation_-failed	DROP COLLATION @1 failed Ошибка в операторе DROP COLLATION @1.
-901	336397277	dsql_create_domain_-failed	CREATE DOMAIN @1 failed Ошибка в операторе CREATE DOMAIN @1.
-901	336397278	dsql_alter_domain_-failed	ALTER DOMAIN @1 failed Ошибка в операторе ALTER DOMAIN @1.
-901	336397279	dsql_drop_domain_failed	DROP DOMAIN @1 failed Ошибка в операторе DROP DOMAIN @1.
-901	336397280	dsql_create_except_-failed	CREATE EXCEPTION @1 failed Ошибка в операторе CREATE EXCEPTION @1.
-901	336397281	dsql_alter_except_-failed	ALTER EXCEPTION @1 failed Ошибка в операторе ALTER EXCEPTION @1.
-901	336397282	dsql_create_alter_-except_failed	CREATE OR ALTER EXCEPTION @1 failed Ошибка в операторе CREATE OR ALTER EXCEPTION @1.
-901	336397283	dsql_recreate_except_-failed	RECREATE EXCEPTION @1 failed Ошибка в операторе RECREATE EXCEPTION @1.
-901	336397284	dsql_drop_except_failed	DROP EXCEPTION @1 failed Ошибка в операторе DROP EXCEPTION @1.
-901	336397285	dsql_create_sequence_-failed	CREATE SEQUENCE @1 failed Ошибка в операторе CREATE SEQUENCE @1.
-901	336397286	dsql_create_table_-failed	CREATE TABLE @1 failed Ошибка в операторе CREATE TABLE @1.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	336397287	dsql_alter_table_failed	ALTER TABLE @1 failed Ошибка в операторе ALTER TABLE @1.
-901	336397288	dsql_drop_table_failed	DROP TABLE @1 failed Ошибка в операторе DROP TABLE @1.
-901	336397289	dsql_recreate_table_- failed	RECREATE TABLE @1 failed Ошибка в операторе RECREATE TABLE @1.
-901	336397290	dsql_create_pack_failed	CREATE PACKAGE @1 failed Ошибка в операторе CREATE PACKAGE @1.
-901	336397291	dsql_alter_pack_failed	ALTER PACKAGE @1 failed Ошибка в операторе ALTER PACKAGE @1.
-901	336397292	dsql_create_alter_pack_- failed	CREATE OR ALTER PACKAGE @1 failed Ошибка в операторе CREATE OR ALTER PACKAGE @1.
-901	336397293	dsql_drop_pack_failed	DROP PACKAGE @1 failed Ошибка в операторе DROP PACKAGE @1.
-901	336397294	dsql_recreate_pack_- failed	RECREATE PACKAGE @1 failed Ошибка в операторе RECREATE PACKAGE @1.
-901	336397295	dsql_create_pack_body_- failed	CREATE PACKAGE BODY @1 failed Ошибка в операторе CREATE PACKAGE BODY @1.
-901	336397296	dsql_drop_pack_body_- failed	DROP PACKAGE BODY @1 failed Ошибка в операторе DROP PACKAGE BODY @1.
-901	336397297	dsql_recreate_pack_- body_failed	RECREATE PACKAGE BODY @1 failed Ошибка в операторе RECREATE PACKAGE BODY @1.
-901	336397298	dsql_create_view_failed	CREATE VIEW @1 failed Ошибка в операторе CREATE VIEW @1.
-901	336397299	dsql_alter_view_failed	ALTER VIEW @1 failed Ошибка в операторе ALTER VIEW @1.
-901	336397300	dsql_create_alter_view_- failed	CREATE OR ALTER VIEW @1 failed Ошибка в операторе CREATE OR ALTER VIEW @1.
-901	336397301	dsql_recreate_view_- failed	RECREATE VIEW @1 failed Ошибка в операторе RECREATE VIEW @1.
-901	336397302	dsql_drop_view_failed	DROP VIEW @1 failed Не используется.
-901	336397303	dsql_drop_sequence_- failed	DROP SEQUENCE @1 failed Ошибка в операторе DROP SEQUENCE @1.
-901	336397304	dsql_recreate_sequence_- failed	RECREATE SEQUENCE @1 failed Ошибка в операторе RECREATE SEQUENCE @1.
-901	336397305	dsql_drop_index_failed	DROP INDEX @1 failed Ошибка в операторе DROP INDEX @1.
-901	336397306	dsql_drop_filter_failed	DROP FILTER @1 failed Ошибка в операторе DROP FILTER @1.
-901	336397307	dsql_drop_shadow_failed	DROP SHADOW @1 failed Ошибка в операторе DROP SHADOW @1.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	336397308	dsql_drop_role_failed	DROP ROLE @1 failed Ошибка в операторе DROP ROLE @1.
-901	336397309	dsql_drop_user_failed	DROP USER @1 failed Ошибка в операторе DROP USER @1.
-901	336397310	dsql_create_role_failed	CREATE ROLE @1 failed Ошибка в операторе CREATE ROLE @1.
-901	336397311	dsql_alter_role_failed	ALTER ROLE @1 failed Не используется.
-901	336397312	dsql_alter_index_failed	ALTER INDEX @1 failed Ошибка в операторе ALTER INDEX @1.
-901	336397313	dsql_alter_database_-failed	ALTER DATABASE failed Ошибка в операторе ALTER DATABASE @1.
-901	336397314	dsql_create_shadow_-failed	CREATE SHADOW @1 failed Ошибка в операторе CREATE SHADOW @1.
-901	336397315	dsql_create_filter_-failed	DECLARE FILTER @1 failed Ошибка в операторе DECLARE FILTER @1.
-901	336397316	dsql_create_index_-failed	CREATE INDEX @1 failed Ошибка в операторе CREATE INDEX @1.
-901	336397317	dsql_create_user_failed	CREATE USER @1 failed Ошибка в операторе CREATE USER @1.
-901	336397318	dsql_alter_user_failed	ALTER USER @1 failed Ошибка в операторе ALTER USER @1.
-901	336397319	dsql_grant_failed	GRANT failed Ошибка в операторе GRANT.
-901	336397320	dsql_revoke_failed	REVOKE failed Ошибка в операторе REVOKE.
-104	336397321	dsql_cte_recursive_-aggregate	Recursive member of CTE cannot use aggregate or window function Рекурсивный член CTE не может использовать агрегатные или оконные функции.
-901	336397322	dsql_mapping_failed	@2 MAPPING @1 failed Ошибка в операторе @2 MAPPING @1.
-901	336397323	dsql_alter_sequence_-failed	ALTER SEQUENCE @1 failed Ошибка в операторе ALTER SEQUENCE @1.
-901	336397324	dsql_create_generator_-failed	CREATE GENERATOR @1 failed Не используется.
-901	336397325	dsql_set_generator_-failed	SET GENERATOR @1 failed Ошибка в операторе SET GENERATOR @1.
-104	336397326	dsql_wlock_simple	WITH LOCK can be used only with a single physical table Предложение WITH LOCK доступно только для выборки данных из одной таблицы.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-104	336397327	dsql_firstskip_rows	FIRST/SKIP cannot be used with OFFSET/FETCH or ROWS Предложения OFFSET и/или FETCH не могут быть объединены с предложениями ROWS или FIRST/SKIP в одном выражении запроса.
-104	336397328	dsql_wlock_aggregates	WITH LOCK cannot be used with aggregates Предложение WITH LOCK нельзя использовать при использовании любых агрегатных функций.
-104	336397329	dsql_wlock_conflict	WITH LOCK cannot be used with @1 Предложение WITH LOCK нельзя использовать с @1.
-901	336397330	dsql_max_exception_-arguments	Number of arguments (@1) exceeds the maximum (@2) number of EXCEPTION USING arguments Количество аргументов (@1) превышает максимальное (@2) число аргументов EXCEPTION USING.
-901	336397331	dsql_string_byte_length	String literal with @1 bytes exceeds the maximum length of @2 bytes Строковый литерал размером @1 байт превышает максимальную длину в @2 байт.
-901	336397332	dsql_string_char_length	String literal with @1 characters exceeds the maximum length of @2 characters for the @3 character set Строковый литерал размером @1 символов превышает максимальную длину в @2 символов для набора символов @3.
-901	336397333	dsql_max_nesting	Too many BEGIN...END nesting. Maximum level is @1 Слишком много вложенных блоков BEGIN...END, максимальный уровень - @1.
-901	336398288	dsql_create_policy_-failed	CREATE POLICY @1 failed Не используется.
-901	336398289	dsql_alter_policy_-failed	ALTER POLICY @1 failed Не используется.
-901	336398290	dsql_drop_policy_failed	DROP POLICY @1 failed Не используется.
-901	336723983	gsec_cant_open_db	unable to open database Не используется.
-901	336723984	gsec_switches_error	error in switch specifications Ошибка в задании переключателя.
-901	336723985	gsec_no_op_spec	no operation specified Не используется.
-901	336723986	gsec_no_usr_name	no user name specified Не используется.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	336723987	gsec_err_add	add record error Не используется.
-901	336723988	gsec_err_modify	modify record error Не используется.
-901	336723989	gsec_err_find_mod	find/modify record error Не используется.
-901	336723990	gsec_err_rec_not_found	record not found for user: @1 Не используется.
-901	336723991	gsec_err_delete	delete record error Не используется.
-901	336723992	gsec_err_find_del	find/delete record error Не используется.
-901	336723996	gsec_err_find_disp	find/display record error Не используется.
-901	336723997	gsec_inv_param	invalid parameter, no switch defined Недопустимый параметр, такой переключатель не определен.
-901	336723998	gsec_op_specified	operation already specified Не используется.
-901	336723999	gsec_pw_specified	password already specified Не используется.
-901	336724000	gsec_uid_specified	uid already specified Не используется.
-901	336724001	gsec_gid_specified	gid already specified Не используется.
-901	336724002	gsec_proj_specified	project already specified Не используется.
-901	336724003	gsec_org_specified	organization already specified Не используется.
-901	336724004	gsec_fname_specified	first name already specified Не используется.
-901	336724005	gsec_mname_specified	middle name already specified Не используется.
-901	336724006	gsec_lname_specified	last name already specified Не используется.
-901	336724008	gsec_inv_switch	invalid switch specified Не используется.
-901	336724009	gsec_amb_switch	ambiguous switch specified Не используется.
-901	336724010	gsec_no_op_specified	no operation specified for parameters Не используется.
-901	336724011	gsec_params_not_allowed	no parameters allowed for this operation Параметры недопустимы для этой операции.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	336724012	gsec_incompat_switch	incompatible switches specified Не используется.
-901	336724044	gsec_inv_username	Invalid user name (maximum 31 bytes allowed) Не используется.
-901	336724045	gsec_inv_pw_length	Warning - maximum 8 significant bytes of password used Не используется.
-901	336724046	gsec_db_specified	database already specified Не используется.
-901	336724047	gsec_db_admin_specified	database administrator name already specified Не используется.
-901	336724048	gsec_db_admin_pw_-specified	database administrator password already specified Не используется.
-901	336724049	gsec_sql_role_specified	SQL role name already specified Не используется.
-901	336920577	gstat_unknown_switch	found unknown switch Не используется.
-901	336920578	gstat_retry	please retry, giving a database name Не используется.
-901	336920579	gstat_wrong_ods	Wrong ODS version, expected @1, encountered @2 Не используется.
-901	336920580	gstat_unexpected_eof	Unexpected end of database file. Не используется.
-901	336920605	gstat_open_err	Can't open database file @1 Не используется.
-901	336920606	gstat_read_err	Can't read a database page Не используется.
-901	336920607	gstat_sysmemex	System memory exhausted Не используется.
-901	336986113	fbsvcmgr_bad_am	Wrong value for access mode Неверное значение для режима доступа.
-901	336986114	fbsvcmgr_bad_wm	Wrong value for write mode Неверное значение для режима записи.
-901	336986115	fbsvcmgr_bad_rs	Wrong value for reserve space Неверное значение для зарезервированного пространства.
-901	336986116	fbsvcmgr_info_err	Unknown tag (@1) in info_svr_db_info block after isc_svc_query() Неизвестный тег (@1) в блоке info_svr_db_info после вызова isc_svc_query().

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	336986117	fbsvcmgr_query_err	Unknown tag (@1) in isc_svc_query() results Неизвестный тег (@1) в результатах isc_svc_query()
-901	336986118	fbsvcmgr_switch_unknown	Unknown switch "@1" Неизвестный параметр командной строки "@1".
-901	336986159	fbsvcmgr_bad_sm	Wrong value for shutdown mode Неверное значение для режима останова.
-901	336986160	fbsvcmgr_fp_open	could not open file @1 Не удается открыть файл @1.
-901	336986161	fbsvcmgr_fp_read	could not read file @1 Не удается прочитать файл @1.
-901	336986162	fbsvcmgr_fp_empty	empty file @1 Пустой файл @1.
-901	336986164	fbsvcmgr_bad_arg	Invalid or missing parameter for switch @1 Недопустимый или отсутствующий параметр для переключателя @1.
-901	337051649	utl_trusted_switch	Switches trusted_user and trusted_role are not supported from command line Переключатели trusted_user и trusted_role не поддерживаются в командной строке.
-901	337117213	nbackup_missing_param	Missing parameter for switch @1 Отсутствует параметр для переключателя @1.
-901	337117214	nbackup_allowed_-switches	Only one of -LOCK, -UNLOCK, -FIXUP, -BACKUP or -RESTORE should be specified Необходимо указать только один из: -LOCK, -UNLOCK, -FIXUP, -BACKUP или -RESTORE.
-901	337117215	nbackup_unknown_param	Unrecognized parameter @1 Нераспознанный параметр @1.
-901	337117216	nbackup_unknown_switch	Unknown switch @1 Неизвестный переключатель @1.
-901	337117217	nbackup_nofetchpw_svc	Fetch password can't be used in service mode Исходный пароль нельзя использовать в сервисном режиме.
-901	337117218	nbackup_pwfile_error	Error working with password file "@1" Ошибка при работе с файлом пароля "@1".
-901	337117219	nbackup_size_with_lock	Switch -SIZE can be used only with -LOCK Переключатель -SIZE может быть использован только вместе в -LOCK.
-901	337117220	nbackup_no_switch	None of -LOCK, -UNLOCK, -FIXUP, -BACKUP or -RESTORE specified Не указана ни одна опция: -LOCK, -UNLOCK, -FIXUP, -BACKUP или -RESTORE.
-901	337117223	nbackup_err_read	IO error reading file: @1 Ошибка ввода-вывода при чтении файла: @1.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	337117224	nbackup_err_write	IO error writing file: @1 Ошибка ввода-вывода при записи в файл: @1.
-901	337117225	nbackup_err_seek	IO error seeking file: @1
-901	337117226	nbackup_err_opendb	Error opening database file: @1 Ошибка открытия файла базы данных: @1.
-901	337117227	nbackup_err_fadvice	Error in posix_fadvise(@1) for database @2 Ошибка в posix_fadvise (@1) для базы данных @ 2.
-901	337117228	nbackup_err_createdb	Error creating database file: @1 Ошибка при создании файла базы данных: @1.
-901	337117229	nbackup_err_openbk	Error opening backup file: @1 Ошибка при открытии файла бэкапа: @1.
-901	337117230	nbackup_err_createbk	Error creating backup file: @1 Ошибка при создании файла бэкапа: @1.
-901	337117231	nbackup_err_eofdb	Unexpected end of database file @1 Неожиданное окончание файла базы данных @1.
-901	337117232	nbackup_fixup_- wrongstate	Database @1 is not in state (@2) to be safely fixed up База данных @1 не в состоянии (@2), чтобы быть безопасно починенной.
-901	337117233	nbackup_err_db	Database error Ошибка базы данных.
-901	337117234	nbackup_userpw_toolong	Username or password is too long Логин или пароль слишком длинный.
-901	337117235	nbackup_lostrec_db	Cannot find record for database "@1" backup level @2 in the backup history Не удается найти запись для базы данных "@1" уровня резервного копирования @2 в истории резервного копирования.
-901	337117236	nbackup_lostguid_db	Internal error. History query returned null SCN or GUID Внутренняя ошибка. Запрос истории копирования базы данных возвратил нулевой системный номер или GUID.
-901	337117237	nbackup_err_eofhdrdb	Unexpected end of file when reading header of database file "@1" (стадия @2) Неожиданный конец файла при чтении заголовка файла базы данных "@1" (этап @2).
-901	337117238	nbackup_db_notlock	Internal error. Database file is not locked. Flags are @1 Внутренняя ошибка. Файл базы данных не заблокирован. Флаги @1.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	337117239	nbackup_lostguid_bk	Internal error. Cannot get backup guid clumplet Внутренняя ошибка. Не удается получить GUID резервной копии.
-901	337117240	nbackup_page_changed	Internal error. Database page @1 had been changed during backup (page SCN=@2, backup SCN=@3) Внутренняя ошибка. Страница базы данных @1 была изменена во время резервного копирования (страница SCN = @2, SCN резервной копии = @3).
-901	337117241	nbackup_dbsize_-inconsistent	Database file size is not a multiple of page size Размер файла базы данных не кратен размеру страницы.
-901	337117242	nbackup_failed_lzbk	Level 0 backup is not restored Резервная копия уровня 0 не восстанавливается.
-901	337117243	nbackup_err_eofhdrbk	Unexpected end of file when reading header of backup file: @1 Неожиданный конец файла при чтении заголовка файла резервной копии: @1.
-901	337117244	nbackup_invalid_incbk	Invalid incremental backup file: @1 Недопустимый инкрементный файл резервной копии: @1.
-901	337117245	nbackup_unsupvers_incbk	Unsupported version @1 of incremental backup file: @2 Неподдерживаемая версия @1 файла инкрементной резервной копии: @2.
-901	337117246	nbackup_invlevel_incbk	Invalid level @1 of incremental backup file: @2, expected @3 Недопустимый уровень @1 инкрементного файла резервной копии: @2, ожидается @3.
-901	337117247	nbackup_wrong_orderbk	Wrong order of backup files or invalid incremental backup file detected, file: @1 Неверный порядок файлов резервных копий или обнаружен недопустимый файл инкрементной резервной копии, файл: @1.
-901	337117248	nbackup_err_eofbk	Unexpected end of backup file: @1 Неожиданный конец файла резервной копии: @1.
-901	337117249	nbackup_err_copy	Error creating database file: @1 via copying from: @2 Ошибка при создании файла базы данных: @1 путем копирования из: @2.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	337117250	nbackup_err_eofhdr_-restdb	Unexpected end of file when reading header of restored database file (stage @1) Неожиданный конец файла при чтении заголовка восстановленного файла базы данных (состояние @1).
-901	337117251	nbackup_lostguid_l0bk	Cannot get backup guid clumplet from L0 backup Не удается получить GUID резервной копии из бэкапа уровня 0.
-901	337117255	nbackup_switchd_-parameter	Wrong parameter @1 for switch -D, need ON or OFF Неверный параметр @1 для переключателя -D, нужен: ON или OFF.
-901	337117257	nbackup_user_stop	Terminated due to user request Завершение, обусловленное запросом пользователя.
-901	337117259	nbackup_deco_parse	Too complex decompress command (> @1 arguments) Слишком сложная команда распаковки (> @1 аргументов).
-901	337118185	nbackup_lostrec_guid_db	Cannot find record for database "@1" backup GUID @2 in the backup history Не удается найти запись для базы данных "@1", GUID резервной копии @2 в истории резервного копирования.
-901	337182750	trace_conflict_acts	conflicting actions "@1" and "@2" found Найдены конфликтующие действия "@1" и "@2".
-901	337182751	trace_act_notfound	action switch not found Переключатель действия не найден.
-901	337182752	trace_switch_once	switch "@1" must be set only once Переключатель "@1" должен быть установлен только один раз.
-901	337182753	trace_param_val_miss	value for switch "@1" is missing Значение для переключателя "@1" отсутствует.
-901	337182754	trace_param_invalid	invalid value ("@1") for switch "@2" Неверное значение ("@1") переключателя "@2".
-901	337182755	trace_switch_unknown	unknown switch "@1" encountered Неизвестный переключатель "@1".
-901	337182756	trace_switch_svc_only	switch "@1" can be used by service only Не используется
-901	337182757	trace_switch_user_only	switch "@1" can be used by interactive user only Переключатель "@1" может использоваться только интерактивным пользователем.

(разрыв таблицы)

(разрыв таблицы)

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	337182758	trace_switch_param_miss	mandatory parameter "@1" for switch "@2" is missing Обязательный параметр "@1" для переключателя "@2" отсутствует.
-901	337182759	trace_param_act_-notcompat	parameter "@1" is incompatible with action "@2" Параметр "@1" несовместим с действием "@2".
-901	337182760	trace_mandatory_switch_-miss	mandatory switch "@1" is missing Обязательный переключатель "@1" отсутствует

Б.2 Коды ошибок SQLSTATE

SQLSTATE предназначен для замены SQLCODE. Последняя в настоящее время устарела и буден удалена будущих версиях.

В блоках обработки ошибок "WHEN ... DO" контекстная переменная SQLSTATE содержит 5 символов SQL-2003 — совместимого кода состояния, переданного оператором, вызвавшим ошибку. Любой код SQLSTATE состоит из двух символов класса (первые два) и трёх символов подкласса. Класс 00 (успешное выполнение), 01 (предупреждение) и 02 (нет данных) представляют собой условия завершения. Каждый код статуса вне этих классов является исключением.

Поскольку классы 00, 01 и 02 не вызывают ошибку, они никогда не будут обнаруживаться в переменной SQLSTATE.

Таблица Б.3 — Классы SQLCLASS

Код SQLCLASS	**Описание**	**Код SQLCLASS**	**Описание**
00	Success	01	Warning
02	No Data	07	Dynamic SQL error
08	Connection Exception	0A	Feature Not Supported
0B	Invalid Transaction Initiation	0L	Invalid Grantor
0P	Invalid Role Specification	0U	Attempt to Assign to Non-Updatable Column
0V	Attempt to Assign to Ordering Column	20	Case Not Found For Case Statement
21	Cardinality Violation	22	Data Exception
23	Integrity Constraint Violation	24	Invalid Cursor State
25	Invalid Transaction State	26	Invalid SQL Statement Name
27	Triggered Data Change Violation	28	Invalid Authorization Specification
2B	Dependent Privilege Descriptors Still Exist	2C	Invalid Character Set Name
2D	Invalid Transaction Termination	2E	Invalid Connection Name
2F	SQL Routine Exception	33	Invalid SQL Descriptor Name

(разрыв таблицы)

(разрыв таблицы)

Код SQLCLASS	Описание	Код SQLCLASS	Описание
34	Invalid Cursor Name	35	Invalid condition number
36	Cursor Sensitivity Exception	38	External Routine Exception
39	External Routine Invocation Exception	3B	Invalid Save Point
3C	Ambiguous Cursor Name	3D	Invalid Catalog Name
3F	Invalid Schema Name	40	Transaction Rollback
42	Syntax Error or Access Violation	44	With Check Option Violation
45	Unhandled user-defined exception	54	Program Limit Exceeded
HY	CLI-specific condition	XX	Internal Error

Таблица Б.4 — Коды ошибок SQLSTATE

Код SQLSTATE	Текст сообщения	Описание
00000	Success	Успешное выполнение
01000	General warning	Общее предупреждение
01001	Cursor operation conflict	Конфликт при операции с курсором
01002	Disconnect error	Ошибка разъединения
01003	NULL value eliminated in set function	Значение NULL устранено в заданной функции
01004	String data, right-truncated	Строковые данные, обрезание справа
01005	Insufficient item descriptor areas	Недостаточно элементов в области дескрипторов
01006	Privilege not revoked	Привилегия не отозвана
01007	Privilege not granted	Привилегия не выдана
01008	Implicit zero-bit padding	Неявное обрезание нулевого бита
01100	Statement reset to unprepared	Оператор сброшен в состояние unprepared
01101	Ongoing transaction has been committed	Текущая транзакция завершена по COMMIT
01102	Ongoing transaction has been rolled back	Текущая транзакция завершена по ROLLED BACK
02000	No data found or no rows affected	Не найдено ни данных, ни строк
07000	Dynamic SQL error	Ошибка DSQL
07001	Wrong number of input parameters	Неверное число входных параметров
07002	Wrong number of output parameters	Неверное число выходных параметров
07003	Cursor specification cannot be executed	Определение курсора не может быть выполнено

(разрыв таблицы)

(разрыв таблицы)

Код SQLSTATE	Текст сообщения	Описание
07004	USING clause required for dynamic parameters	Для динамического параметра требуется предложение USING
07005	Prepared statement not a cursor specification	Подготовленный оператор не является курсор - специфичным
07006	Restricted data type attribute violation	Исключение по причине запрещенного типа данных для атрибута
07007	USING clause required for result fields	Для возвращаемого поля требуется предложение USING
07008	Invalid descriptor count	Неверный счетчик дескрипторов
07009	Invalid descriptor index	Неверный индекс дескриптора
08001	Client unable to establish connection	Клиент не может установить соединение
08002	Connection name in use	Имя соединения уже используется
08003	Connection does not exist	Соединение не существует
08004	Server rejected the connection	Сервер отклонил подключение
08006	Connection failure	Ошибка при подключении
08007	Transaction resolution unknown	Неизвестно разрешение транзакции
0A000	Feature not supported	Возможность не поддерживается
0B000	Invalid transaction initiation	Неверная инициализация транзакции
0L000	Invalid grantor	Неверный грантор (тот, кто дает привилегии)
0P000	Invalid role specification	Неверная спецификация роли
0U000	Attempt to assign to non-updatable column	Попытка присвоения не обновляемому столбцу
0V000	Attempt to assign to Ordering Column	Попытка присвоения сортируемому столбцу
20000	Case not found for case statement	Не обнаружено вариантов для предложения CASE
21000	Cardinality violation	Нарушение количества элементов
21S01	Insert value list does not match column list	Список вставляемых значений не соответствует списку столбцов
21S02	Degree of derived table does not match column list	Состояние производной таблицы не соответствует списку столбцов
22000	Data exception	Исключения данных
22001	String data, right truncation	Строковые данные, усечены справа
22002	Null value, no indicator parameter	Значение NULL, параметр не обозначен
22003	Numeric value out of range	Числовое значение вышло за предел допустимого
22004	Null value not allowed	Значение NULL не допустимо
22005	Error in assignment	Ошибка присваивания

(разрыв таблицы)

(разрыв таблицы)

Код SQLSTATE	Текст сообщения	Описание
22006	Null value in field reference	Значение NULL в поле ссылки
22007	Invalid datetime format	Неверный формат даты/времени
22008	Datetime field overflow	Переполнение в поле даты/времени
22009	Invalid time zone displacement value	Недопустимое значение смещения часового пояса
2200A	Null value in reference target	Значение NULL в целевой ссылке
2200B	Escape character conflict	Конфликт символа экранирования
2200C	Invalid use of escape character	Неверное использование символа экранирования
2200D	Invalid escape octet	Неверный октет для управляющего символа
2200E	Null value in array target	Значение NULL в массиве назначения
2200F	Zero-length character string	Нулевая длина строки символов
2200G	Most specific type mismatch	Наиболее определенное несоответствие типа
22010	Invalid indicator parameter value	Неверный индикатор значения параметра
22011	Substring error	Ошибка подстроки
22012	Division by zero	Деление на ноль
22014	Invalid update value	Неверное значение в операции update
22015	Interval field overflow	Переполнение интервала в поле
22018	Invalid character value for cast	Неверный символ для преобразования типов
22019	Invalid escape character	Неверный символ экранирования
2201B	Invalid regular expression	Неверное регулярное выражение
2201C	Null row not permitted in table	Запись, содержащая NULL, не допустима для таблицы
22020	Invalid limit value	Неверное значение лимита
22021	Character not in repertoire	Символ вне диапазона
22022	Indicator overflow	Переполнение индикатора
22023	Invalid parameter value	Неверное значение параметра
22024	Character string not properly terminated	Символьная строка имеет некорректный замыкающий символ
22025	Invalid escape sequence	Неверная управляющая последовательность
22026	String data, length mismatch	Строковые данные, длина неверная
22027	Trim error	Ошибка операции TRIM
22028	Row already exists	Строка уже существует
2202D	Null instance used in mutator function	NULL экземпляр используется для функции, изменяющей значение объекта

(разрыв таблицы)

(разрыв таблицы)

Код SQLSTATE	Текст сообщения	Описание
2202E	Array element error	Ошибка элемента массива
2202F	Array data, right truncation	Данные массива, обрезание справа
23000	Integrity constraint violation	Нарушение ограничения целостности
24000	Invalid cursor state	Неверное состояние курсора
24504	The cursor identified in the UPDATE, DELETE, SET, or GET statement is not positioned on a row	Курсор, определенный для UPDATE, DELETE, SET или GET операции, не позиционирован по строке
25000	Invalid transaction state	Неверное состояние транзакции
25S01	Transaction state	Состояние транзакции
25S02	Transaction is still active	Транзакция до сих пор активная
25S03	Transaction is rolled back	Транзакция откачена
26000	Invalid SQL statement name	Неверное имя SQL выражения
27000	Triggered data change violation	Ошибки при изменении данных триггером
28000	Invalid authorization specification	Неверная спецификация авторизации
2B000	Dependent privilege descriptors still exist	Зависимые описания привилегий еще существуют
2C000	Invalid character set name	Неверное имя набора символов
2D000	Invalid transaction termination	Неверное завершение транзакции
2E000	Invalid connection name	Неверное имя соединения
2F000	SQL routine exception	Процедурное исключение SQL
2F002	Modifying SQL-data not permitted	Для модификации SQL-данных нет доступа
2F003	Prohibited SQL-statement attempted	Встретился запрещенный SQL запрос
2F004	Reading SQL-data not permitted	Нет доступа на чтение SQL-данных
2F005	Function executed no return statement	Исполняемая функция не имеет возвращаемого выражения
33000	Invalid SQL descriptor name	Недопустимое имя дескриптора SQL
34000	Invalid cursor name	Неверное имя курсора
35000	Invalid condition number	Неверный номер условия
36001	Request rejected	Запрос отказан
36002	Request failed	Запрос ошибочный
38000	External routine exception	Ошибка внешней процедуры
39000	External routine invocation exception	Ошибка вызова внешней процедуры
3B000	Invalid save point	Неверная точка сохранения
3C000	Ambiguous cursor name	Имя курсора неоднозначное
3D000	Invalid catalog name	Неверное имя каталога

(разрыв таблицы)

(разрыв таблицы)

Код SQLSTATE	Текст сообщения	Описание
3D001	Catalog name not found	Каталог с таким именем не обнаружен
3F000	Invalid schema name	Неверное имя схемы
40000	Ongoing transaction has been rolled back	Текущая транзакция была откачена
40001	Serialization failure	Отказ сериализации
40002	Transaction integrity constraint violation	Нарушение условия целостности транзакции
40003	Statement completion unknown	Неизвестно состояние завершения транзакции
42000	Syntax error or access violation	Синтаксическая ошибка или ошибка доступа
42702	Ambiguous column reference	Неоднозначная ссылка на столбец
42725	Ambiguous function reference	Неоднозначная ссылка на функцию
42818	The operands of an operator or function are not compatible	Операнды оператора или функции являются не совместимыми
42S01	Base table or view already exists	Таблица или представление уже существует
42S02	Base table or view not found	Таблица или представление не найдено
42S11	Index already exists	Индекс уже существует
42S12	Index not found	Индекс не найден
42S21	Column already exists	Столбец уже существует
42S22	Column not found	Столбец не найден
44000	WITH CHECK OPTION violation	Нарушение опции WITH CHECK
45000	Unhandled user-defined exception	Необработанное исключение, определенное пользователем
54000	Program limit exceeded	Превышены ограничения программы
54001	Statement too complex	Выражение слишком сложное
54011	Too many columns	Слишком много столбцов
54023	Too many arguments	Слишком много аргументов
HY000	CLI-specific condition	CLI-специфическое условие
HY001	Memory allocation error	Ошибка выделения памяти
HY003	Invalid data type in application descriptor	Неверный тип данных в дескрипторе приложения
HY004	Invalid data type	Неверный тип данных
HY007	Associated statement is not prepared	Связанный оператор не подготовлен
HY008	Operation canceled	Операция отменена
HY009	Invalid use of null pointer	Неправильное использование нулевого указателя
HY010	Function sequence error	Ошибка последовательности функций

(разрыв таблицы)

(разрыв таблицы)

Код SQLSTATE	Текст сообщения	Описание
HY011	Attribute cannot be set now	Атрибут не может быть установлен сейчас
HY012	Invalid transaction operation code	Неверный код транзакции операции
HY013	Memory management error	Ошибка управления памятью
HY014	Limit on the number of handles exceeded	Достигнут лимит числа указателей
HY015	No cursor name available	Недоступен курсор без имени
HY016	Cannot modify an implementation row descriptor	Невозможно изменить реализацию дескриптора строки
HY017	Invalid use of an automatically allocated descriptor handle	Неверное использование автоматически выделяемого дескриптора указателей
HY018	Server declined the cancellation request	Сервер отклонил запрос на отмену
HY019	Non-string data cannot be sent in pieces	Не строковые данные не могут быть отправлены по частям
HY020	Attempt to concatenate a null value	Попытка конкатенации значения NULL
HY021	Inconsistent descriptor information	Противоречивая информация о дескрипторе
HY024	Invalid attribute value	Неверное значение атрибута
HY055	Non-string data cannot be used with string routine	Не строковые данные не могут быть использованы со строковой процедурой
HY090	Invalid string length or buffer length	Неверная длина строки или длина буфера
HY091	Invalid descriptor field identifier	Неверный дескриптор идентификатора поля
HY092	Invalid attribute identifier	Неверный идентификатор атрибута
HY095	Invalid FunctionId specified	Неверное указание ID функции
HY096	Invalid information type	Неверный тип информации
HY097	Column type out of range	Тип столбца вне диапазона
HY098	Scope out of range	Определение вне диапазона
HY099	Nullable type out of range	Типы с допустимыми NULL вне диапазона
HY100	Uniqueness option type out of range	Тип опции "уникальность" вне диапазона
HY101	Accuracy option type out of range	Тип опции "точность" вне диапазона
HY103	Invalid retrieval code	Неверный код поиска
HY104	Invalid LengthPrecision value	Неверное значение длина/точность
HY105	Invalid parameter type	Неверный тип параметра
HY106	Invalid fetch orientation	Неверное направление для fetch
HY107	Row value out of range	Значение строки вне диапазона

(разрыв таблицы)

(разрыв таблицы)

Код SQLSTATE	Текст сообщения	Описание
HY109	Invalid cursor position	Неверная позиция курсора
HY110	Invalid driver completion	Неверный код завершения драйвера
HY111	Invalid bookmark value	Неверное значение метки bookmark
HYC00	Optional feature not implemented	Опциональная функция не реализована
HYT00	Timeout expired	Достигнут тайм-аут
HYT01	Connection timeout expired	Достигнут тайм-аут соединения
XX000	Internal error	Внутренняя ошибка
XX001	Data corrupted	Данные разрушены
XX002	Index corrupted	Индекс разрушен

Приложение B Описание системных таблиц

При первоначальном создании базы данных система управления базами данных создает множество системных таблиц. В системных таблицах хранятся метаданные — описания всех объектов базы данных.

Список системных таблиц в алфавитном порядке представлен в таблице B.1.

Таблица B.1 — Список системных таблиц «РЕД База Данных»

Таблица	Содержание
RDB$AUTH_MAPPING	Сведения об отображении объектов безопасности
RDB$BACKUP_HISTORY	Хранит историю копирования базы данных
RDB$CHARACTER_SETS	Доступные в базе данных наборы символов
RDB$CHECK_CONSTRAINTS	Соответствие имен триггеров именам ограничений, связанных с характеристиками NOT NULL, ограничениями CHECK и предложениями ON UPDATE и ON DELETE в ограничениях внешнего ключа
RDB$COLLATIONS	Порядки сортировки для всех наборов символов
RDB$CONFIG	Настройки конфигурации текущей базы данных для текущего подключения
RDB$DATABASE	Основные данные о базе данных
RDB$DB_CREATORS	Содержит сведения о пользователях имеющих права на создание базы данных. Используется только в том случае, если текущая база данных назначена как база данных безопасности.
RDB$CONSTANTS	Хранит информацию о константах.
RDB$DEPENDENCIES	Сведения о зависимостях между объектами базы данных
RDB$EXCEPTIONS	Пользовательские исключения базы данных
RDB$FIELDS	Характеристики столбцов и доменов, как системных, так и созданных пользователем
RDB$FIELD_DIMENSIONS	Размерности столбцов, являющихся массивами
RDB$FILES	Сведения о вторичных файлах и файлах оперативных копий
RDB$FILTERS	Данные о BLOB — фильтрах
RDB$FORMATS	Данные об изменениях таблиц
RDB$FUNCTIONS	Сведения о внешних или хранимых функциях
RDB$FUNCTION_ARGUMENTS	Характеристики параметров внешних или хранимых функций
RDB$GENERATORS	Сведения о генераторах (последовательностях)
RDB$INDEX_SEGMENTS	Сегменты и позиции индексов
RDB$INDICES	Определение индексов базы данных (созданных пользователем или системой)
RDB$KEYWORDS	Ключевые слова
RDB$LOG_FILES	В настоящей версии не используется
RDB$PACKAGES	Сведения о PSQL пакетах
RDB$PAGES	Сведения о страницах базы данных
RDB$PROCEDURE_PARAMETERS	Параметры хранимых процедур

(разрыв таблицы)

(разрыв таблицы)

Таблица	Содержание
RDB$PROCEDURES	Описания хранимых процедур
RDB$REF_CONSTRAINTS	Описания именованных ограничений базы данных (внешних ключей)
RDB$RELATION_CONSTRAINTS	Описание всех ограничений на уровне таблиц
RDB$RELATION_FIELDS	Характеристики столбцов таблиц
RDB$RELATIONS	Заголовки таблиц и представлений
RDB$ROLES	Определение ролей
RDB$SECURITY_CLASSES	Списки управления доступом
RDB$TIME_ZONES	Списки часовых поясов поддерживаемых сервером
RDB$TRANSACTIONS	Состояние транзакций при обращении к нескольким базам данных
RDB$TRIGGER_MESSAGES	Сообщения триггеров
RDB$TRIGGERS	Описания триггеров
RDB$TYPES	Описание перечислимых типов данных
RDB$PUBLICATIONS	Хранит публикации репликации, заданные в базе данных
RDB$PUBLICATION_TABLES	Хранит имена таблиц, которые реплицируются в рамках публикации
RDB$USER_PRIVILEGES	Полномочия пользователей системы
RDB$VIEW_RELATIONS	Описывает представления. Не используется в настоящей версии

RDB$AUTH_MAPPING

Таблица содержит сведения об отображении объектов безопасности.

Идентификатор столбца	Тип данных	Описание
RDB$MAP_NAME	CHAR(63)	Имя отображения.
RDB$MAP_USING	CHAR(1)	Является ли аутентификация общесерверной (S) или обычной (P).
RDB$MAP_PLUGIN	CHAR(63)	Имя плагина аутентификации, из которого происходит отображение.
RDB$MAP_DB	CHAR(63)	Имя базы данных, в которой прошла аутентификация. Из неё происходит отображение.
RDB$MAP_FROM_TYPE	CHAR(63)	Тип объекта, который будет отображён.
RDB$MAP_FROM	CHAR(255)	Имя объекта, из которого будет произведено отображение.
RDB$MAP_TO_TYPE	SMALLINT	Тип объекта, в который будет произведено отображение: 0 — USER; 1 — ROLE.
RDB$MAP_TO	CHAR(63)	Наименование объекта, в который будет произведено отображение (имя пользователя или роли).

(разрыв таблицы)

(разрыв таблицы)

Идентификатор столбца	Тип данных	Описание
RDB$SYSTEM_FLAG	SMALLINT	0 — определён пользователем; 1 —определён в системе.
RDB$DESCRIPTION	BLOB TEXT	Произвольное текстовое описание порядка сортировки.

RDB$BACKUP_HISTORY

Таблица хранит историю копирования базы данных при помощи утилиты `nbackup`.

Идентификатор столбца	Тип данных	Описание
RDB$BACKUP_ID	INTEGER	Присваиваемый системой идентификатор.
RDB$TIMESTAMP	TIMESTAMP WITH TIME ZONE	Дата и время выполнения копирования.
RDB$BACKUP_LEVEL	INTEGER	Уровень копирования.
RDB$GUID	CHAR(38)	Уникальный идентификатор.
RDB$SCN	INTEGER	Системный номер.
RDB$FILE_NAME	VARCHAR(255)	Полный путь и имя файла копии.

RDB$CHARACTER_SETS

Содержит наборы символов, доступные в базе данных.

Идентификатор столбца	Тип данных	Описание
RDB$CHARACTER_SET_NAME	CHAR(63)	Имя набора символов.
RDB$FORM_OF_USE	CHAR(63)	Не используется.
RDB$NUMBER_OF_CHARACTERS	INTEGER	Количество символов в наборе. Для существующих наборов символов не используется.
RDB$DEFAULT_COLLATE_NAME	CHAR(63)	Имя порядка сортировки по умолчанию для набора символов.
RDB$CHARACTER_SET_ID	SMALLINT	Уникальный идентификатор набора символов.
RDB$SYSTEM_FLAG	SMALLINT	Системный флаг: имеет значение 1, если набор символов был определен в системе при создании базы данных; значение 0 для набора символов, определенного пользователем.
RDB$DESCRIPTION	BLOB TEXT	Произвольное текстовое описание набора символов.
RDB$FUNCTION_NAME	CHAR(63)	Имя внешней функции для наборов символов, определенных пользователем, доступ к которым осуществляется через внешнюю функцию.
RDB$BYTES_PER_CHARACTER	SMALLINT	Количество байтов для представления одного символа.

(разрыв таблицы)

(разрыв таблицы)

Идентификатор столбца	Тип данных	Описание
RDB$SECURITY_CLASS	CHAR(63)	Может ссылаться на класс безопасности, определённый в таблице RDB$SECURITY_CLASSES для применения ограничений управления доступом для всех пользователей этого набора символов.
RDB$OWNER_NAME	CHAR(63)	Имя пользователя — владельца (создателя) набора символов.

RDB$CHECK_CONSTRAINTS

Описывает соответствие имен триггеров именам ограничений, связанных с характеристиками NOT NULL, ограничениями CHECK и предложениями ON UPDATE, ON DELETE в ограничениях внешнего ключа.

Идентификатор столбца	Тип данных	Описание
RDB$CONSTRAINT_NAME	CHAR(63)	Имя ограничения. Задается пользователем или автоматически генерируется системой.
RDB$TRIGGER_NAME	CHAR(63)	Для ограничения CHECK — это имя триггера, который поддерживает данное ограничение. Для ограничения NOT NULL — это имя столбца, к которому применяется ограничение. Для ограничения внешнего ключа — это имя триггера, который поддерживает предложения ON UPDATE, ON DELETE.

RDB$COLLATIONS

Порядки сортировки для наборов символов.

Идентификатор столбца	Тип данных	Описание
RDB$COLLATION_NAME	CHAR(63)	Имя порядка сортировки.
RDB$COLLATION_ID	SMALLINT	Идентификатор порядка сортировки. Вместе с идентификатором набора символов является уникальным идентификатором порядка сортировки.
RDB$CHARACTER_SET_ID	SMALLINT	Идентификатор набора символов. Вместе с идентификатором порядка сортировки является уникальным идентификатором.
RDB$COLLATION_ATTRIBUTES	SMALLINT	Атрибуты сортировки. Представляет собой битовую маску, где 1-й бит показывает учитывать ли конечные пробелы при сравнении (0 - NO PAD; 1 - PAD SPACE); 2-й бит показывает является ли сравнение чувствительным к регистру символов (0 - CASE SENSITIVE, 1 - CASE INSENSITIVE); 3-й бит показывает будет ли сравнение чувствительным к акцентам (0 - ACCENT SENSITIVE, 1 - ACCENT INSENSITIVE). Таким образом, значение 5 означает, что сравнение не является чувствительным к конечным пробелам и к акцентированным буквам.

(разрыв таблицы)

(разрыв таблицы)

Идентификатор столбца	Тип данных	Описание
RDB$SYSTEM_FLAG	SMALLINT	Признак: определен пользователем — значение 0; определен в системе — значение 1.
RDB$DESCRIPTION	BLOB TEXT	Произвольное текстовое описание порядка сортировки.
RDB$FUNCTION_NAME	CHAR(63)	В настоящий момент не используется.
RDB$BASE_COLLATION_NAME	CHAR(63)	Имя базового порядка сортировки для данного порядка сортировки.
RDB$SPECIFIC_ATTRIBUTES	BLOB TEXT	Описание особых атрибутов.
RDB$SECURITY_CLASS	CHAR(63)	Может ссылаться на класс безопасности, определённый в таблице RDB$SECURITY_CLASSES для применения ограничений управления доступом для всех пользователей этой сортировки.
RDB$OWNER_NAME	CHAR(63)	Имя пользователя — владельца (создателя) сортировки.

RDB$CONFIG

Содержит сведения о параметры конфигурации текущей базы данных для текущего подключения. Доступна только администраторам, другие пользователи не видят строк этой таблицы.

Идентификатор столбца	Тип данных	Описание
RDB$CONFIG_ID	INTEGER	Идентификатор строки.
RDB$CONFIG_NAME	VARCHAR(63)	Название параметра конфигурации.
RDB$CONFIG_VALUE	VARCHAR(255)	Значение параметра конфигурации.
RDB$CONFIG_DEFAULT	VARCHAR(255)	Значение по умолчанию для параметра конфигурации.
RDB$CONFIG_IS_SET	BOOLEAN	TRUE, если значение настроено, FALSE, установлено значение по умолчанию.
RDB$CONFIG_SOURCE	VARCHAR(255)	Имя файла конфигурации (относительно корневого каталога), из которого был взят этот параметр, или специальное значение DPB, если параметр был указан клиентским приложением через API.

RDB$CONSTANTS

Хранит информацию о константах.

Идентификатор столбца	Тип данных	Описание
RDB$CONSTANT_NAME	CHAR(63)	Имя константы
RDB$PACKAGE_NAME	CHAR(63)	Пакет константы, для которой описывается зависимость.

(разрыв таблицы)

(разрыв таблицы)

Идентификатор столбца	Тип данных	Описание
RDB$FIELD_SOURCE	CHAR(63)	Содержит имя домена (определенного пользователем или созданного автоматически системой), на котором основывается данный столбец.
RDB$PRIVATE_FLAG	SMALLINT	Указывает, определина константа в теле пакета (значение `true`) или в заголовке пакета (значеение `false`).
RDB$CONSTANT_BLR	BLOB BLR	Значение константы, сохранённое в виде `blr`.
RDB$CONSTANT_SOURCE	BLOB TEXT	Определение коснтанты в исходном (SQL) виде.

RDB$DATABASE

Основные данные о базе данных. Содержит только одну запись.

Идентификатор столбца	Тип данных	Описание
RDB$DESCRIPTION	BLOB TEXT	Текст примечания базы данных.
RDB$RELATION_ID	SMALLINT	Количество таблиц и представлений в базе данных.
RDB$SECURITY_CLASS	CHAR(63)	Класс безопасности, определенный в RDB$SECURITY_CLASSES, для обращения к общим для базы данных ограничениям доступа.
RDB$CHARACTER_SET_NAME	CHAR(63)	Имя набора символов по умолчанию для базы данных, установленного в предложении DEFAULT CHARACTER SET при создании базы данных. NULL — набор символов NONE.
RDB$LINGER	INTEGER	Количество секунд "задержки" (установленной оператором `alter database set linger`) до закрытия последнего соединения базы данных (в SuperServer). Если задержка не установлена, то содержит NULL.
RDB$SQL_SECURITY	BOOLEAN	Определяет в контексте какого пользователя будет выполняться объект базы данных (процедура, функция, пакет, триггер, таблица) по умолчанию. Если установлен в FALSE, то объект выполняется с правами вызвавшего его пользователя. Иначе объект выполняется с правами его владельца (создателя)

RDB$DB_CREATORS

Содержит сведения о пользователях имеющих права на создание базы данных. Используется только в том случае, если текущая база данных назначена как база данных безопасности.

Идентификатор столбца	Тип данных	Описание
RDB$USER	CHAR(63)	Имя пользователя или роли, которому даны полномочия на создание базы данных.

(разрыв таблицы)

(разрыв таблицы)

Идентификатор столбца	Тип данных	Описание
RDB$USER_TYPE	SMALLINT	Тип пользователя: 8 — пользователь; 13 — роль.

RDB$DEPENDENCIES

Сведения о зависимостях между объектами базы данных.

Идентификатор столбца	Тип данных	Описание
RDB$DEPENDENT_NAME	CHAR(63)	Имя представления, процедуры, триггера, ограничения CHECK или вычисляемого столбца, для которого описывается зависимость.
RDB$DEPENDED_ON_NAME	CHAR(63)	Объект, зависящий от описываемого объекта — таблица, на которую ссылается представление, процедура, триггер, ограничение CHECK или вычисляемый столбец.
RDB$FIELD_NAME	CHAR(63)	Имя столбца в зависимой таблице, на который ссылается представление, процедура, триггер, ограничение CHECK или вычисляемый столбец.
RDB$DEPENDENT_TYPE	SMALLINT	Идентифицирует тип описываемого объекта: 0 – таблица, 1 – представление, 2 – триггер, 3 – вычисляемый столбец, 4 – ограничение CHECK, 5 – процедура, 6 – выражение для индекса, 7 – исключение, 8 – пользователь, 9 – столбец, 10 – индекс, 15 — хранимая функция, 18 — заголовок пакета, 19 — тело пакета.

(разрыв таблицы)

(разрыв таблицы)

Идентификатор столбца	Тип данных	Описание
RDB$DEPENDED_ON_TYPE	SMALLINT	Идентифицирует тип зависимого объекта: 0 – таблица (или её столбец), 1 – представление, 2 – триггер, 3 – вычисляемый столбец, 4 – ограничение CHECK, 5 – процедура, 6 – выражение для индекса, 7 – исключение, 8 – пользователь, 9 – столбец, 10 – индекс, 14 — генератор (последовательность), 15 — UDF или хранимая функция, 17 — сортировка, 18 — заголовок пакета, 19 — тело пакета.
RDB$PACKAGE_NAME	CHAR(63)	Пакет процедуры или функции, для которой описывается зависимость.

RDB$EXCEPTIONS

Пользовательские исключения базы данных.

Идентификатор столбца	Тип данных	Описание
RDB$EXCEPTION_NAME	CHAR(63)	Имя пользовательского исключения.
RDB$EXCEPTION_NUMBER	INTEGER	Назначенный системой уникальный номер исключения.
RDB$MESSAGE	VARCHAR(1023)	Текст сообщения в исключении.
RDB$DESCRIPTION	BLOB TEXT	Произвольное текстовое описание исключения.
RDB$SYSTEM_FLAG	SMALLINT	Признак: определено пользователем = 0; определено системой = 1 или выше.
RDB$SECURITY_CLASS	CHAR(63)	Может ссылаться на класс безопасности, определённый в таблице RDB$SECURITY_CLASSES для применения ограничений управления доступом для всех пользователей этого исключения.
RDB$OWNER_NAME	CHAR(63)	Имя пользователя — владельца (создателя) исключения.

RDB$FIELDS

Характеристики столбцов и доменов, как системных, так и созданных пользователем.

Идентификатор столбца	Тип данных	Описание
RDB$FIELD_NAME	CHAR(63)	Уникальное имя домена, созданного пользователем, или домена, автоматически построенного системой для столбца таблицы. Во втором случае имя будет начинаться с символов 'RDB$'.
RDB$QUERY_NAME	CHAR(63)	Не используется.
RDB$VALIDATION_BLR	BLOB BLR	Двоичное представление (BLR) выражения SQL, задающее проверку значения CHECK у домена.
RDB$VALIDATION_SOURCE	BLOB TEXT	Оригинальный исходный текст на языке SQL, задающий проверку значения CHECK.
RDB$COMPUTED_BLR	BLOB BLR	Двоичное представление (BLR) выражения SQL, которое используется сервером базы данных для вычисления при обращении к столбцу COMPUTED BY.
RDB$COMPUTED_SOURCE	BLOB TEXT	Оригинальный исходный текст выражения, которое определяет столбец COMPUTED BY.
RDB$DEFAULT_VALUE	BLOB BLR	Значение по умолчанию в двоичном виде BLR.
RDB$DEFAULT_SOURCE	BLOB TEXT	Значение по умолчанию в исходном виде на языке SQL.
RDB$FIELD_LENGTH	SMALLINT	Размер столбца в байтах. FLOAT, DATE, TIME, INTEGER занимают 4 байта. DOUBLE PRECISION, BIGINT, TIMESTAMP и идентификатор BLOB — 8 байтов. Для типов данных CHAR и VARCHAR столбец задает максимальное количество байтов, указанное при объявлении строкового домена (столбца).
RDB$FIELD_SCALE	SMALLINT	Отрицательное число задает масштаб для столбцов DECIMAL и NUMERIC — количество дробных знаков после десятичной точки.
RDB$FIELD_TYPE	SMALLINT	Код типа данных для столбца: 7 = SMALLINT, 8 = INTEGER, 10 = FLOAT, 12 = DATE, 13 = TIME, 14 = CHAR, 16 = BIGINT, 23 = BOOLEAN, 24 = DECFLOAT(16), 25 = DECFLOAT(32), 26 = INT128, 27 = DOUBLE PRECISION, 28 = TIME WITH TIME ZONE,

(разрыв таблицы)

(разрыв таблицы)

Идентификатор столбца	Тип данных	Описание
RDB$FIELD_TYPE	SMALLINT	29 = TIMESTAMP WITH WITH TIME ZONE, 35 = TIMESTAMP, 37 = VARCHAR, 261 = BLOB. Коды для DECIMAL и NUMERIC имеют тот же размер, что и целые типы, используемые для их хранения.
RDB$FIELD_SUB_TYPE	SMALLINT	Для типа данных BLOB задает подтип: 0 – не определен, 1 – текст, 2 – BLR, 3 – список управления доступом, 4 – резервируется для дальнейшего использования, 5 – кодированное описание метаданных таблицы, 6 – описание транзакции к нескольким базам данных, которая не завершилась нормально. Для типа данных CHAR задает: 0 – неопределенные данные, 1 – фиксированные двоичные данные. Для целочисленных типов данных (SMALLINT, INTEGER, BIGINT) и чисел с фиксированной точкой (NUMERIC, DECIMAL) задает конкретный тип данных: 0 или NULL – тип данных соответствует значению в поле RDB$FIELD_TYPE, 1 – NUMERIC, 2 – DECIMAL.
RDB$MISSING_VALUE	BLOB BLR	Не используется.
RDB$MISSING_SOURCE	BLOB TEXT	Не используется.
RDB$DESCRIPTION	BLOB TEXT	Произвольный текст комментария для домена (столбца таблицы).
RDB$SYSTEM_FLAG	SMALLINT	Признак: значение 1 — домен, автоматически созданный системой, значение 0 — домен определен пользователем.
RDB$QUERY_HEADER	BLOB TEXT	Не используется.
RDB$SEGMENT_LENGTH	SMALLINT	Для столбцов BLOB задает длину буфера BLOB в байтах. Для остальных типов данных содержит NULL.
RDB$EDIT_STRING	VARCHAR(127)	Не используется.
RDB$EXTERNAL_LENGTH	SMALLINT	Длина столбца в байтах, если он входит в состав внешней таблицы. Всегда NULL для обычных таблиц.

(разрыв таблицы)

(разрыв таблицы)

Идентификатор столбца	Тип данных	Описание
RDB$EXTERNAL_SCALE	SMALLINT	Показатель степени для столбца целого типа данных во внешней таблице; задается степенью 10, на которую умножается целое.
RDB$EXTERNAL_TYPE	SMALLINT	Тип данных поля, как он представляется во внешней таблице. 7 = SMALLINT, 8 = INTEGER, 10 = FLOAT, 12 = DATE, 13 = TIME, 14 = CHAR, 16 = BIGINT, 23 = BOOLEAN, 27 = DOUBLE PRECISION, 35 = TIMESTAMP, 37 = VARCHAR. Коды для DECIMAL и NUMERIC имеют тот же размер, что и целые типы, используемые для их хранения.
RDB$DIMENSIONS	SMALLINT	Задает количество размерностей массива, если столбец был определен как массив. Для столбцов, не являющихся массивами, всегда NULL.
RDB$NULL_FLAG	SMALLINT	Указывает, может ли столбец принимать пустое значение (в поле будет значение NULL) или не может (в поле будет содержаться значение 1).
RDB$CHARACTER_LENGTH	SMALLINT	Длина столбцов CHAR или VARCHAR в символах (не в байтах).
RDB$COLLATION_ID	SMALLINT	Идентификатор порядка сортировки для символьного столбца или домена. Если не задан, значением поля будет 0.
RDB$CHARACTER_SET_ID	SMALLINT	Идентификатора набора символов для символьного столбца, столбца BLOB или домена.
RDB$FIELD_PRECISION	SMALLINT	Указывает общее количество цифр для числового типа данных с фиксированной точкой (DECIMAL и NUMERIC). Для целочисленных типов данных значением является 0, для всех остальных типов данных — NULL.
RDB$SECURITY_CLASS	CHAR(63)	Может ссылаться на класс безопасности, определённый в таблице RDB$SECURITY_CLASSES для применения ограничений управления доступом для всех пользователей этого домена.
RDB$OWNER_NAME	CHAR(63)	Имя пользователя – владельца (создателя) домена.

RDB$FIELD_DIMENSIONS

Размерности столбцов, являющихся массивами.

Идентификатор столбца	Тип данных	Описание
RDB$FIELD_NAME	CHAR(63)	Имя столбца, являющегося массивом. Должно содержаться в поле RDB$FIELD_NAME таблицы RDB$FIELDS.
RDB$DIMENSION	SMALLINT	Определяет одну размерность столбца массива. Нумерация размерностей начинается с 0.
RDB$LOWER_BOUND	INTEGER	Нижняя граница этой размерности.
RDB$UPPER_BOUND	INTEGER	Верхняя граница описываемой размерности.

RDB$FILES

Сведения о вторичных файлах и файлах оперативных копий.

Идентификатор столбца	Тип данных	Описание
RDB$FILE_NAME	VARCHAR(255)	Полный путь к файлу и имя вторичного файла базы данных в многофайловой базе данных или файла оперативной копии.
RDB$FILE_SEQUENCE	SMALLINT	Порядковый номер вторичного файла в последовательности или номер файла копии в наборе оперативных копий.
RDB$FILE_START	INTEGER	Начальный номер страницы вторичного файла или файла оперативной копии.
RDB$FILE_LENGTH	INTEGER	Длина файла в страницах базы данных.
RDB$FILE_FLAGS	SMALLINT	Для внутреннего использования.
RDB$SHADOW_NUMBER	SMALLINT	Номер набора оперативных копий. Если строка описывает вторичный файл базы данных, то значением поля будет NULL или 0.

RDB$FILTERS

Содержит данные о BLOB-фильтрах.

Идентификатор столбца	Тип данных	Описание
RDB$FUNCTION_NAME	CHAR(63)	Уникальное имя фильтра BLOB.
RDB$DESCRIPTION	BLOB TEXT	Написанная пользователем документация о фильтре BLOB и используемых двух подтипах.
RDB$MODULE_NAME	VARCHAR(255)	Имя динамической библиотеки / совместно используемого объекта, где расположен код фильтра BLOB.
RDB$ENTRYPOINT	CHAR(255)	Точка входа в библиотеке фильтров для этого фильтра BLOB.
RDB$INPUT_SUB_TYPE	SMALLINT	Подтип BLOB для преобразуемых данных.

(разрыв таблицы)

(разрыв таблицы)

Идентификатор столбца	Тип данных	Описание
RDB$OUTPUT_SUB_TYPE	SMALLINT	Подтип BLOB, в который преобразуются входные данные.
RDB$SYSTEM_FLAG	SMALLINT	Признак: внешне определенный фильтр (т.е. определенный пользователем = значение 0, внутренне определенный = значение 1 или более)
RDB$SECURITY_CLASS	CHAR(63)	Может ссылаться на класс безопасности, определённый в таблице RDB$SECURITY_CLASSES для применения ограничений управления доступом для всех пользователей этого BLOB фильтра.
RDB$OWNER_NAME	CHAR(63)	Имя пользователя – владельца (создателя) BLOB фильтра.

RDB$FORMATS

Данные об изменениях таблиц. Каждый раз, когда таблица изменяется, таблица получает новый номер формата. Когда номер формата любой таблицы достигает 255, вся база данных становится недоступной для работы с ней. Тогда нужно выполнить резервное копирование, восстановить эту копию и продолжить работу с заново созданной базой данных.

Идентификатор столбца	Тип данных	Описание
RDB$RELATION_ID	SMALLINT	Идентификатор таблицы или представления.
RDB$FORMAT	SMALLINT	Идентификатор формата таблицы. Форматов может быть до 255.
RDB$DESCRIPTOR	BLOB FORMAT	Отображение в виде BLOB столбцов и характеристик данных на момент, когда была создана запись формата.

RDB$FUNCTIONS

Сведения о внешних или хранимых функциях.

Идентификатор столбца	Тип данных	Описание
RDB$FUNCTION_NAME	CHAR(63)	Уникальное имя внешней функции.
RDB$FUNCTION_TYPE	SMALLINT	В настоящий момент не используется.
RDB$QUERY_NAME	CHAR(63)	В настоящий момент не используется.
RDB$DESCRIPTION	BLOB TEXT	Произвольный текст комментария к внешней функции.
RDB$MODULE_NAME	VARCHAR(255)	Имя модуля (библиотеки DLL), где расположен код функции.
RDB$ENTRYPOINT	CHAR(255)	Имя точки входа в библиотеке, где находится эта функция.
RDB$RETURN_ARGUMENT	SMALLINT	Номер позиции возвращаемого аргумента в списке параметров, соответствующем входным аргументам.

(разрыв таблицы)

(разрыв таблицы)

Идентификатор столбца	Тип данных	Описание
RDB$SYSTEM_FLAG	SMALLINT	Признак определения функции: определенная пользователем = 1, определенная системой = 0.
RDB$ENGINE_NAME	CHAR(63)	Имя движка для использования внешних функций. Обычно UDR.
RDB$PACKAGE_NAME	CHAR(63)	Имя пакета, если функция является упакованной.
RDB$PRIVATE_FLAG	SMALLINT	Для неупакованных хранимых функций всегда NULL, для упакованных 0 — если функция описана в заголовке пакета и 1 — если функция описана или реализована только в теле пакета (не описана в заголовке).
RDB$FUNCTION_SOURCE	BLOB TEXT	Исходный код функции на языке SQL.
RDB$FUNCTION_ID	SMALLINT	Уникальный идентификатор функции.
RDB$FUNCTION_BLR	BLOB BLR	Двоичное представление (BLR) кода функции.
RDB$VALID_BLR	SMALLINT	Указывает, остается ли текст хранимой функции корректным после последнего изменения функции при помощи оператора ALTER FUNCTION.
RDB$DEBUG_INFO	BLOB	Содержит отладочную информацию о переменных, используемых в хранимой функции.
RDB$SECURITY_CLASS	CHAR(63)	Может ссылаться на класс безопасности, определённый в таблице RDB$SECURITY_CLASSES для применения ограничений управления доступом.
RDB$OWNER_NAME	CHAR(63)	Имя пользователя – владельца (создателя) функции.
RDB$LEGACY_FLAG	SMALLINT	Признак legacy стиля функции. 1 — если функция описана в legacy стиле (DECLARE EXTERNAL FUNCTION), в противном случае 0 (CREATE FUNCTION).
RDB$DETERMINISTIC_FLAG	SMALLINT	Флаг детерминистической функции. 1 - если функция детерминистическая (DETERMINISTIC), в противном случае - 0.
RDB$SQL_SECURITY	BOOLEAN	Определяет в контексте какого пользователя будет выполняться функция. Если установлен в FALSE, то функция выполняется с правами вызвавшего его пользователя. Иначе функция выполняется с правами его владельца (создателя)

RDB$FUNCTION_ARGUMENTS

Характеристики параметров внешних или хранимых функций.

Идентификатор столбца	Тип данных	Описание
RDB$FUNCTION_NAME	CHAR(63)	Уникальное имя внешней функции.
RDB$ARGUMENT_POSITION	SMALLINT	Позиция аргумента в списке аргументов.

(разрыв таблицы)

(разрыв таблицы)

Идентификатор столбца	Тип данных	Описание
RDB$MECHANISM	SMALLINT	Механизм передачи параметра для Legacy функций: передается ли аргумент по значению (значение столбца 0), по ссылке (значение 1), через дескриптор (значение 2) или через дескриптор BLOB (значение 3).
RDB$FIELD_TYPE	SMALLINT	Число, задающее код типа данных, определенного для столбца: 7 = SMALLINT, 8 = INTEGER, 12 = DATE, 13 = TIME, 14 = CHAR, 16 = BIGINT, 23 = BOOLEAN; 27 = DOUBLE PRECISION, 35 = TIMESTAMP, 37 = VARCHAR, 40 = CSTRING (строка, завершаемая нулем), 45 = blob_id, 261 = BLOB.
RDB$FIELD_SCALE	SMALLINT	Масштаб для целого числа или аргумента с фиксированной точкой. Это показатель числа 10.
RDB$FIELD_LENGTH	SMALLINT	Длина аргумента в байтах. BOOLEAN = 1, SMALLINT = 2, INTEGER = 4, DATE = 4, TIME = 4, BIGINT = 8, DOUBLE PRECISION = 8, TIMESTAMP = 8, blob_id = 8.
RDB$FIELD_SUB_TYPE	SMALLINT	Для аргумента типа данных BLOB задает подтип BLOB.
RDB$CHARACTER_SET_ID	SMALLINT	Идентификатор набора символов для символьного аргумента.
RDB$FIELD_PRECISION	SMALLINT	Количество цифр точности, допустимой для типа данных аргумента.
RDB$CHARACTER_LENGTH	SMALLINT	Длина аргумента CHAR или VARCHAR в символах (но не в байтах).
RDB$PACKAGE_NAME	CHAR(63)	Имя пакета функции (если функция упакованная), в которой используется параметр.
RDB$ARGUMENT_NAME	CHAR(63)	Имя параметра.

(разрыв таблицы)

(разрыв таблицы)

Идентификатор столбца	Тип данных	Описание
RDB$FIELD_SOURCE	CHAR(63)	Имя домена, созданного пользователем (при использовании ссылки на домен вместо типа), или домена, автоматически построенного системой для параметра функции. Во втором случае имя будет начинаться с символов 'RDB$'.
RDB$DEFAULT_VALUE	BLOB BLR	Значение по умолчанию на языке BLR.
RDB$DEFAULT_SOURCE	BLOB TEXT	Значение по умолчанию в исходном виде на языке SQL.
RDB$COLLATION_ID	SMALLINT	Идентификатор используемого порядка сортировки для символьного параметра.
RDB$NULL_FLAG	SMALLINT	Признак допустимости пустого значения NULL.
RDB$ARGUMENT_MECHANISM	SMALLINT	Механизм передачи параметра для не Legacy функций: передается ли аргумент по значению (значение столбца 0), по ссылке (значение 1), через дескриптор (значение 2) или через дескриптор BLOB (значение 3).
RDB$FIELD_NAME	CHAR(63)	Имя столбца, на которое ссылается параметр с помощью предложения TYPE OF COLUMN.
RDB$RELATION_NAME	CHAR(63)	Имя таблицы, на которую ссылается параметр с помощью предложения TYPE OF COLUMN.
RDB$SYSTEM_FLAG	SMALLINT	Указывает, является ли параметр определённым системой (значение 1 и выше) или пользователем (значение 0).
RDB$DESCRIPTION	BLOB TEXT	Текст произвольного примечания к параметру.

RDB$GENERATORS

Сведения о генераторах (последовательностях).

Идентификатор столбца	Тип данных	Описание
RDB$GENERATOR_NAME	CHAR(63)	Уникальное имя генератора.
RDB$GENERATOR_ID	SMALLINT	Назначаемый системой уникальный идентификатор для генератора.
RDB$SYSTEM_FLAG	SMALLINT	Признак: значение 0 — генератор определен пользователем, значение 1 или выше — определен системой, 6 — внутренний генератор для identity столбца.
RDB$DESCRIPTION	BLOB TEXT	Произвольный текст примечания к генератору.
RDB$SECURITY_CLASS	CHAR(63)	Может ссылаться на класс безопасности, определённый в таблице RDB$SECURITY_CLASSES для применения ограничений управления доступом.
RDB$OWNER_NAME	CHAR(63)	Имя пользователя – владельца (создателя) генератора.

(разрыв таблицы)

(разрыв таблицы)

Идентификатор столбца	Тип данных	Описание
RDB$INITIAL_VALUE	BIGINT	Хранит начальное значение генератора или значение генератора, установленное при предыдущем рестарте (WITH RESTART).
RDB$GENERATOR_INCREMENT	INTEGER	Шаг приращения генератора при использовании оператора NEXT VALUE FOR.

RDB$INDEX_SEGMENTS

Сегменты и позиции индексов. Таблица описывает все столбцы таблицы, входящие в состав конкретного индекса. Для каждого столбца индекса создается отдельная строка в данной таблице.

Идентификатор столбца	Тип данных	Описание
RDB$INDEX_NAME	CHAR(63)	Имя индекса, к которому относится данный сегмент. Должно соответствовать главной записи в системной таблице RDB$INDICES.
RDB$FIELD_NAME	CHAR(63)	Имя одного из столбцов, входящего в состав индекса. Должно соответствовать значению в столбце RDB$FIELD_NAME в таблице RDB$RELATION_FIELDS.
RDB$FIELD_POSITION	SMALLINT	Позиция столбца в индексе. Нумерация начинается с нуля.
RDB$STATISTICS	DOUBLE PRECISION	Селективность индекса по данному столбцу.

RDB$INDICES

Определение индексов базы данных (созданных пользователем или системой). Описывает каждый индекс, созданный пользователем или системой. Для каждого столбца таблицы, входящего в состав индекса, присутствует строка системной таблицы RDB$INDEX_SEGMENTS, где описываются характеристики столбца индекса.

Идентификатор столбца	Тип данных	Описание
RDB$INDEX_NAME	CHAR(63)	Уникальное имя индекса, заданное пользователем или автоматически сгенерированное системой.
RDB$RELATION_NAME	CHAR(63)	Имя таблицы, к которой применяется индекс. Соответствует RDB$RELATION_NAME в строке таблицы RDB$RELATIONS.
RDB$INDEX_ID	SMALLINT	Внутренний (системный) идентификатор индекса.
RDB$UNIQUE_FLAG	SMALLINT	Указывает, является ли индекс уникальным: 1 — уникальный, 0 — не уникальный.
RDB$DESCRIPTION	BLOB TEXT	Произвольный текст комментария к индексу.
RDB$SEGMENT_COUNT	SMALLINT	Количество сегментов (столбцов) в индексе.

(разрыв таблицы)

(разрыв таблицы)

Идентификатор столбца	Тип данных	Описание
RDB$INDEX_INACTIVE	SMALLINT	Указывает, является ли в настоящий момент индекс активным: 1 — неактивный, 0 — активный.
RDB$INDEX_TYPE	SMALLINT	Упорядоченность индекса: 1 - по убыванию значений столбцов 0 - по возрастанию значений столбцов.
RDB$FOREIGN_KEY	CHAR(63)	Имя ассоциированного ограничения внешнего ключа, если существует.
RDB$SYSTEM_FLAG	SMALLINT	Указывает, является ли индекс определенным системой (значение 1 или выше) или пользователем (значение 0).
RDB$EXPRESSION_BLR	BLOB BLR	Выражение, записанное на языке двоичного представления (BLR). Будет использовано для вычисления во время выполнения, когда будут реализованы индексы выражений.
RDB$EXPRESSION_SOURCE	BLOB TEXT	Исходный текст выражения. Будет использовано, когда будут реализованы индексы выражений.
RDB$STATISTICS	DOUBLE PRECISION	Хранит самую последнюю селективность индекса, вычисленную при помощи оператора SET STATISTICS.
RDB$TABLESPACE_NAME	CHAR (63)	Наименование табличного пространства.
RDB$CONDITION_SOURCE	BLOB TEXT	Исходный код выражения для ограничения набора индексируемых записей
RDB$CONDITION_BLR	BLOB BLR	Выражение в формате BLR. Используется для ограничения набора индексируемых записей.

RDB$JOBS

Сведения о заданиях планировщика.

Идентификатор столбца	Тип данных	Описание
RDB$JOB_NAME	CHAR(63)	Имя задания
RDB$JOB_ID	INTEGER	ID задания
RDB$JOB_SOURCE	BLOB TEXT	Код задания в текстовом виде
RDB$JOB_BLR	BLOB BINARY	Скомпилированный BLR для задания
RDB$DESCRIPTION	BLOB TEXT	Содержит описание задания
RDB$OWNER_NAME	CHAR(63)	Имя пользователя, создавшего задание

(разрыв таблицы)

(разрыв таблицы)

Идентификатор столбца	Тип данных	Описание
RDB$JOB_INACTIVE	SMALLINT	Имеет два значения: 1 - задание не будет запускаться по расписанию 0 - задание будет запускаться
RDB$JOB_TYPE	SMALLINT	Имеет два значения: 0 — обычное PSQL задание 1 — задание с командой ОС
RDB$JOB_SCHEDULE	VARBINARY(64)	Строка в формате cron, задающая расписание
RDB$START_DATE	TIMESTAMP WITH TIME ZONE	Дата и время начала выполнения задания
RDB$END_DATE	TIMESTAMP WITH TIME ZONE	Дата и время окончания выполнения задания
RDB$DATABASE	VARCHAR(255)	Имя базы данных, для которой создано задание

RDB$JOBS_LOG

События, связанные с заданиями.

Идентификатор столбца	Тип данных	Описание
RDB$TIMESTAMP	TIMESTAMP WITH TIME ZONE	Время произошедшего события
RDB$JOB_NAME	CHAR(63)	Имя задания
RDB$JOB_ID	INTEGER	ID задания
RDB$EVENT	VARCHAR(32)	Регистрируются следующие типы событий: RUN_START - начало выполнения задания; RUN_FINISH - завершение выполнения задания; RUN_ERROR - ошибка во время выполнения задания.
RDB$MESSAGE	VARCHAR(1023)	Текст ошибки

RDB$KEYWORDS

Содержит информацию о ключевых словах сервера.

Идентификатор столбца	Тип данных	Описание
RDB$KEYWORD_NAME	VARCHAR(63)	Ключевое слово.

(разрыв таблицы)

(разрыв таблицы)

Идентификатор столбца	Тип данных	Описание
RDB$KEYWORD_RESERVED	BOOLEAN	Зарезервировано ли ключевое слово.

RDB$LOG_FILES

В настоящей версии не используется.

Идентификатор столбца	Тип данных	Описание
RDB$FILE_NAME	VARCHAR(255)	Не используется.
RDB$FILE_SEQUENCE	SMALLINT	Не используется.
RDB$FILE_LENGTH	INTEGER	Не используется.
RDB$FILE_PARTITIONS	SMALLINT	Не используется.
RDB$FILE_P_OFFSET	INTEGER	Не используется.
RDB$FILE_FLAGS	SMALLINT	Не используется.

RDB$PACKAGES

В таблице содержатся сведения о PSQL пакетах.

Идентификатор столбца	Тип данных	Описание
RDB$PACKAGE_NAME	CHAR(63)	Уникальное имя пакета.
RDB$PACKAGE_HEADER_SOURCE	BLOB TEXT	Исходный код заголовка пакета на языке SQL.
RDB$PACKAGE_BODY_SOURCE	BLOB TEXT	Исходный код тела пакета на языке SQL.
RDB$VALID_BODY_FLAG	SMALLINT	Указывает, остаётся ли текст тела пакета корректным после последнего изменения заголовка пакета или его пересоздания.
RDB$SECURITY_CLASS	CHAR(63)	Может указывать на класс безопасности, определённый в системной таблице RDB$SECURITY_CLASSES, для применения ограничений управления доступом.
RDB$OWNER_NAME	CHAR(63)	Имя пользователя – владельца (создателя) пакета.
RDB$SYSTEM_FLAG	SMALLINT	Указывает, что пакет определён пользователем (значение 0) или системой (значение 1 или выше).
RDB$DESCRIPTION	BLOB TEXT	Произвольный текст примечания к пакету.
RDB$SQL_SECURITY	BOOLEAN	Определяет в контексте какого пользователя будут выполняться процедуры и функции. Если установлен в FALSE, то они выполняется с правами вызвавшего его пользователя. Иначе они выполняется с правами его владельца (создателя).

RDB$PAGES

Сведения о страницах базы данных.

Идентификатор столбца	Тип данных	Описание
RDB$PAGE_NUMBER	INTEGER	Уникальный номер физически созданной страницы базы данных.
RDB$RELATION_ID	SMALLINT	Идентификатор таблицы, для которой выделена эта страница.
RDB$PAGE_SEQUENCE	INTEGER	Последовательный номер страницы по отношению к другим страницам, выделенным для данной таблицы.
RDB$PAGE_TYPE	SMALLINT	Описывает тип страницы. Для системного использования.

RDB$PROCEDURE_PARAMETERS

Описывает параметры хранимых процедур.

Идентификатор столбца	Тип данных	Описание
RDB$PARAMETER_NAME	CHAR(63)	Имя параметра.
RDB$PROCEDURE_NAME	CHAR(63)	Имя процедуры, в которой используется параметр.
RDB$PARAMETER_NUMBER	SMALLINT	Последовательный номер параметра.
RDB$PARAMETER_TYPE	SMALLINT	Указывает, является ли параметр входным (значение 0) или выходным (значение 1).
RDB$FIELD_SOURCE	CHAR(63)	Сгенерированное системой уникальное глобальное имя столбца.
RDB$DESCRIPTION	BLOB TEXT	Текст произвольного примечания к параметру.
RDB$SYSTEM_FLAG	SMALLINT	Указывает, является ли параметр определенным системой (значение 1 и выше) или пользователем (значение 0).
RDB$DEFAULT_VALUE	BLOB BLR	Значение по умолчанию на языке BLR.
RDB$DEFAULT_SOURCE	BLOB TEXT	Значение по умолчанию в исходном виде на языке SQL.
RDB$COLLATION_ID	SMALLINT	Идентификатор используемого порядка сортировки для символьного параметра.
RDB$NULL_FLAG	SMALLINT	Признак допустимости пустого значения NULL.
RDB$PARAMETER_MECHANISM	SMALLINT	Признак — передается ли параметр по значению (значение столбца 0), по ссылке (значение 1), через дескриптор (значение 2) или через дескриптор BLOB (значение 3).
RDB$FIELD_NAME	CHAR(63)	Имя столбца, на которое ссылается параметр с помощью предложения TYPE OF COLUMN.
RDB$RELATION_NAME	CHAR(63)	Имя таблицы, на которую ссылается параметр с помощью предложения TYPE OF COLUMN.

(разрыв таблицы)

(разрыв таблицы)

Идентификатор столбца	Тип данных	Описание
RDB$PACKAGE_NAME	CHAR(63)	Имя пакета процедуры (если процедура упакованная), в которой используется параметр.

RDB$PROCEDURES

Описывает хранимые процедуры.

Идентификатор столбца	Тип данных	Описание
RDB$PROCEDURE_NAME	CHAR(63)	Имя хранимой процедуры.
RDB$PROCEDURE_ID	SMALLINT	Уникальный идентификатор процедуры.
RDB$PROCEDURE_INPUTS	SMALLINT	Указывает количество входных параметров или их отсутствие (значение NULL).
RDB$PROCEDURE_OUTPUTS	SMALLINT	Указывает количество выходных параметров или их отсутствие (значение NULL).
RDB$DESCRIPTION	BLOB TEXT	Произвольный текст примечания к процедуре.
RDB$PROCEDURE_SOURCE	BLOB TEXT	Исходный код процедуры на языке SQL.
RDB$PROCEDURE_BLR	BLOB BLR	Двоичное представление (BLR) кода процедуры.
RDB$SECURITY_CLASS	CHAR(63)	Может указывать на класс безопасности, определенный в системной таблице RDB$SECURITY_CLASSES, для применения ограничений управления доступом.
RDB$OWNER_NAME	CHAR(63)	Имя пользователя — владельца (создателя) процедуры.
RDB$RUNTIME	BLOB	Описание метаданных процедуры. Внутреннее использование для оптимизации.
RDB$SYSTEM_FLAG	SMALLINT	Указывает, процедура определена пользователем (значение 0) или системой (значение 1 или выше).
RDB$PROCEDURE_TYPE	SMALLINT	Тип процедуры: 1 — хранимая процедура выбора (содержит в своем составе оператор SUSPEND), 2 — выполняемая хранимая процедура.
RDB$VALID_BLR	SMALLINT	Указывает, остается ли текст хранимой процедуры корректным после последнего изменения процедуры при помощи оператора ALTER PROCEDURE.
RDB$DEBUG_INFO	BLOB	Содержит отладочную информацию о переменных, используемых в хранимой процедуре.
RDB$ENGINE_NAME	CHAR(63)	Имя движка для использования внешних процедур. Обычно UDR.
RDB$ENTRYPOINT	CHAR(255)	Имя точки входа в библиотеке, где находится эта процедура.
RDB$PACKAGE_NAME	CHAR(63)	Имя пакета, если процедура является упакованной.

(разрыв таблицы)

(разрыв таблицы)

Идентификатор столбца	Тип данных	Описание
RDB$PRIVATE_FLAG	SMALLINT	Для неупакованных хранимых процедур всегда NULL, для упакованных 0 — если процедура описана в заголовке пакета и 1 — если процедура описана или реализована только в теле пакета (не описана в заголовке)
RDB$SQL_SECURITY	BOOLEAN	Определяет в контексте какого пользователя будет выполняться процедура. Если установлен в FALSE, то процедура выполняется с правами вызвавшего его пользователя. Иначе процедура выполняется с правами его владельца (создателя)

RDB$REF_CONSTRAINTS

Описания именованных ограничений базы данных (внешних ключей).

Идентификатор столбца	Тип данных	Описание
RDB$CONSTRAINT_NAME	CHAR(63)	Имя ограничения внешнего ключа. Задается пользователем или автоматически генерируется системой.
RDB$CONST_NAME_UQ	CHAR(63)	Имя ограничения первичного или уникального ключа, на которое ссылается предложение REFERENCES в данном ограничении.
RDB$MATCH_OPTION	CHAR(7)	Не используется. Текущим значением является FULL во всех случаях.
RDB$UPDATE_RULE	CHAR(11)	Действия по ссылочной целостности, применимые к данному внешнему ключу, когда изменяется первичный (уникальный) ключ родительской таблицы: RESTRICT, NO ACTION, CASCADE, SET NULL, SET DEFAULT.
RDB$DELETE_RULE	CHAR(11)	Действия по ссылочной целостности, применимые к данному внешнему ключу, когда удаляется первичный (уникальный) ключ родительской таблицы: RESTRICT, NO ACTION, CASCADE, SET NULL, SET DEFAULT.

RDB$RELATION_CONSTRAINTS

Описание всех ограничений на уровне таблиц: первичного, уникального, внешнего ключей, ограничений CHECK, NOT NULL.

Идентификатор столбца	Тип данных	Описание
RDB$CONSTRAINT_NAME	CHAR(63)	Имя ограничения на уровне таблицы, заданное пользователем или автоматически присвоенное системой.
RDB$CONSTRAINT_TYPE	CHAR(11)	Содержит название ограничения: PRIMARY KEY, UNIQUE, FOREIGN KEY, CHECK, NOT NULL.

(разрыв таблицы)

(разрыв таблицы)

Идентификатор столбца	Тип данных	Описание
RDB$RELATION_NAME	CHAR(63)	Имя таблицы, к которой применяется это ограничение.
RDB$DEFERRABLE	CHAR(3)	В настоящий момент во всех случаях NO.
RDB$INITIALLY_DEFERRED	CHAR(3)	В настоящий момент во всех случаях NO.
RDB$INDEX_NAME	CHAR(63)	Имя индекса, который поддерживает это ограничение (содержит NULL, если ограничением является CHECK или NOT NULL).

RDB$RELATION_FIELDS

Характеристики столбцов таблиц и представлений.

Идентификатор столбца	Тип данных	Описание
RDB$FIELD_NAME	CHAR(63)	Имя столбца.
RDB$RELATION_NAME	CHAR(63)	Имя таблицы (представления), где присутствует описываемый столбец.
RDB$FIELD_SOURCE	CHAR(63)	Содержит имя домена (определенного пользователем или созданного автоматически системой), на котором основывается данный столбец.
RDB$QUERY_NAME	CHAR(63)	В настоящей версии системы не используется.
RDB$BASE_FIELD	CHAR(63)	Только для представления. Имя столбца из базовой таблицы.
RDB$EDIT_STRING	VARCHAR(127)	Не используется.
RDB$FIELD_POSITION	SMALLINT	Позиция столбца в таблице или представлении. Нумерация начинается с 0.
RDB$QUERY_HEADER	BLOB TEXT	Не используется.
RDB$UPDATE_FLAG	SMALLINT	Указывает, является ли столбец обычным столбцом (значение 1) или вычисляемым (значение 0).
RDB$FIELD_ID	SMALLINT	В настоящей версии системы в точности соответствует значению в столбце RDB$FIELD_POSITION.
RDB$VIEW_CONTEXT	SMALLINT	Для столбца представления это внутренний идентификатор базовой таблицы, откуда приходит это поле.
RDB$DESCRIPTION	BLOB TEXT	Примечание к столбцу таблицы или представления.
RDB$DEFAULT_VALUE	BLOB BLR	Записанное в двоичном виде (BLR) значение по умолчанию — предложение DEFAULT, если оно присутствует при описании столбца таблицы (представления).
RDB$SYSTEM_FLAG	SMALLINT	Указывает, определено пользователем (значение 0) или системой (значение 1 или выше).
RDB$SECURITY_CLASS	CHAR(63)	Может ссылаться на класс безопасности, определенный в RDB$SECURITY_CLASSES для применения ограничений управления доступом для всех пользователей этого столбца.

(разрыв таблицы)

(разрыв таблицы)

Идентификатор столбца	Тип данных	Описание
RDB$COMPLEX_NAME	CHAR(63)	Не используется.
RDB$NULL_FLAG	SMALLINT	Указывает, допускает ли столбец значения NULL (значение NULL) или не допускает (значение 1).
RDB$DEFAULT_SOURCE	BLOB TEXT	Исходный текст предложения DEFAULT, если присутствует.
RDB$COLLATION_ID	SMALLINT	Идентификатор последовательности сортировки в составе набора символов для столбца не по умолчанию.
RDB$GENERATOR_NAME	CHAR(63)	Имя внутреннего генератора для реализации identity столбца.
RDB$IDENTITY_TYPE	SMALLINT	Указывает тип identity столбца: 0 - GENERATED ALWAYS, 1 - GENERATED BY DEFAULT, NULL - не является столбцом идентификации.
RDB$TABLESPACE_NAME	CHAR(63)	Наименование табличного пространства.

RDB$RELATIONS

Хранит некоторые характеристики таблиц и представлений.

Идентификатор столбца	Тип данных	Описание
RDB$VIEW_BLR	BLOB BLR	Для представления содержит на языке BLR спецификации запроса. Для таблицы в поле содержится NULL.
RDB$VIEW_SOURCE	BLOB TEXT	Для представления содержит оригинальный исходный текст запроса на языке SQL (включая пользовательские комментарии). Для таблицы в поле содержится NULL.
RDB$DESCRIPTION	BLOB TEXT	Произвольный текст примечания к таблице (представлению).
RDB$RELATION_ID	SMALLINT	Внутренний идентификатор таблицы (представления).
RDB$SYSTEM_FLAG	SMALLINT	Указывает, создана ли таблица (представление) пользователем (значение 0) или системой (значение 1 или выше).
RDB$DBKEY_LENGTH	SMALLINT	Общая длина ключа. Для таблицы это 8 байтов. Для представления это 8, умноженное на количество таблиц, на которые ссылается представление.
RDB$FORMAT	SMALLINT	Внутреннее использование.
RDB$FIELD_ID	SMALLINT	Количество столбцов в таблице (представлении).
RDB$RELATION_NAME	CHAR(63)	Имя таблицы или представления.

(разрыв таблицы)

(разрыв таблицы)

Идентификатор столбца	Тип данных	Описание
RDB$SECURITY_CLASS	CHAR(63)	Может ссылаться на класс безопасности, определенный в таблице RDB$SECURITY_CLASSES для применения ограничений управления доступом для всех пользователей этой таблицы (представления).
RDB$EXTERNAL_FILE	VARCHAR(255)	Полный путь к внешнему файлу данных, если таблица описана с предложением EXTERNAL FILE.
RDB$RUNTIME	BLOB	Описание метаданных таблицы. Внутреннее использование для оптимизации.
RDB$EXTERNAL_DESCRIPTION	BLOB	Произвольное примечание к внешнему файлу таблицы.
RDB$OWNER_NAME	CHAR(63)	Имя пользователя — владельца (создателя) таблицы или представления.
RDB$DEFAULT_CLASS	CHAR(63)	Класс безопасности по умолчанию. Применяется, когда новый столбец добавляется в таблицу.
RDB$FLAGS	SMALLINT	Внутренние флаги.
RDB$RELATION_TYPE	SMALLINT	Тип описываемого объекта: 0 — системная таблица или таблица, созданная пользователем, 1 — представление, 2 — внешняя таблица, 3 — виртуальная таблица (таблица мониторинга MON$, псевдотаблицы безопасности SEC$), 4 — GTT уровня соединения (PRESERVE ROWS), 5 — GTT уровня транзакции (DELETE ROWS).
RDB$SQL_SECURITY	BOOLEAN	Определяет в контексте какого пользователя будет использоваться таблица. Если установлен в FALSE, то таблица будет использоваться с правами вызвавшего его пользователя. Иначе таблица будет использоваться с правами его владельца (создателя)
RDB$ADAPTER	CHAR(7)	Адаптер для внешней таблицы для чтения двоичных трейсов
RDB$TABLESPACE_NAME	CHAR(63)	Наименование табличного пространства.
RDB$POINTER_PAGE	INTEGER	Номер первой страницы указателей таблицы.
RDB$ROOT_PAGE	INTEGER	Номер страницы корневых индексов таблицы.

RDB$ROLES

Определение ролей.

Идентификатор столбца	Тип данных	Описание
RDB$ROLE_NAME	CHAR(63)	Имя роли.
RDB$OWNER_NAME	CHAR(63)	Имя пользователя-владельца роли.
RDB$DESCRIPTION	BLOB TEXT	Произвольный текст примечания к роли.
RDB$SYSTEM_FLAG	SMALLINT	Системный флаг.
RDB$SECURITY_CLASS	CHAR(63)	Может ссылаться на класс безопасности, определённый в таблице RDB$SECURITY_CLASSES для применения ограничений управления доступом для всех пользователей этой роли
RDB$SYSTEM_PRIVILEGES	BINARY(8)	Флаги системных привилегий

RDB$SECURITY_CLASSES

Списки управления доступом.

Идентификатор столбца	Тип данных	Описание
RDB$SECURITY_CLASS	CHAR(63)	Имя класса безопасности.
RDB$ACL	BLOB ACL	Список управления доступом, связанный с классом безопасности. Перечисляет пользователей и их полномочия.
RDB$DESCRIPTION	BLOB TEXT	Произвольный текст примечания к классу безопасности.

RDB$TABLESPACES

Идентификатор столбца	Тип данных	Описание
RDB$TABLESPACE_ID	INTEGER	Идентификатор табличного пространства.
RDB$TABLESPACE_NAME	CHAR(63)	Наименование табличного пространства.
RDB$SECURITY_CLASS	CHAR (63)	Может ссылаться на класс безопасности, определённый в таблице RDB$SECURITY_CLASSES для применения ограничений управления доступом для всех пользователей этого табличного пространства.
RDB$SYSTEM_FLAG	SMALLINT	Указывает, что пакет определён пользователем (значение 0) или системой (значение 1 или выше).
RDB$DESCRIPTION	BLOB TEXT	Произвольное текстовое описание табличного пространства.
RDB$OWNER_NAME	CHAR (63)	Имя пользователя — владельца (создателя) табличного пространства.
RDB$FILE_NAME	VARCHAR (255)	Полный путь к табличному пространству.
RDB$OFFLINE	BOOLEAN	Не используется.
RDB$READ_ONLY	BOOLEAN	Не используется.

RDB$TIME_ZONES

Виртуальная таблица со списком часовых поясов поддерживаемых сервером.

Идентификатор столбца	Тип данных	Описание
RDB$TIME_ZONE_ID	INTEGER	Идентификатор часового пояса.
RDB$TIME_ZONE_NAME	CHAR(63)	Наименование часового пояса.

RDB$TRANSACTIONS

Состояние транзакций при обращении к нескольким базам данных.

Идентификатор столбца	Тип данных	Описание
RDB$TRANSACTION_ID	INTEGER	Уникальный идентификатор отслеживаемой транзакции.
RDB$TRANSACTION_STATE	SMALLINT	Состояние транзакции: зависшая (значение 0), подтвержденная (значение 1), отмененная (значение 2).
RDB$TIMESTAMP	TIMESTAMP WITH TIME ZONE	Не используется.
RDB$TRANSACTION_DESCRIPTION	BLOB	Описывает подготовленную транзакцию к нескольким базам данных. Используется в случае потери соединения, которое не может быть восстановлено.

RDB$TRIGGER_MESSAGES

Сообщения триггеров.

Идентификатор столбца	Тип данных	Описание
RDB$TRIGGER_NAME	CHAR(63)	Имя триггера, с которым связано данное сообщение.
RDB$MESSAGE_NUMBER	SMALLINT	Номер сообщения в пределах одного триггера (от 1 до максимум 32,767).
RDB$MESSAGE	VARCHAR(1023)	Текст сообщения триггера.

RDB$TRIGGERS

Описания триггеров.

Идентификатор столбца	Тип данных	Описание
RDB$TRIGGER_NAME	CHAR(63)	Имя триггера.

(разрыв таблицы)

(разрыв таблицы)

Идентификатор столбца	Тип данных	Описание
RDB$RELATION_NAME	CHAR(63)	Имя таблицы или представления, для которого используется триггер. Если триггер применяется не к событию таблицы, а к событию базы данных, то в этом поле находится NULL.
RDB$TRIGGER_SEQUENCE	SMALLINT	Последовательность (позиция) триггера. Ноль обычно означает, что последовательность не задана.
RDB$TRIGGER_TYPE	BIGINT	Событие, на которое вызывается триггер. Значение высчитывается как битовая маска. Наиболее распространенные значения: 1 — before insert, 2 — after insert, 3 — before update, 4 — after update, 5 — before delete, 6 — after delete, 17 — before insert or update, 18 — after insert or update, 25 — before insert or delete, 26 — after insert or delete, 27 — before update or delete, 28 — after update or delete, 113- before insert or update or delete 114- after insert or update or delete, 8192 — on connect, 8193 — on disconnect, 8194 — on transaction start, 8195 — on transaction commit, 8196 — on transaction rollback. Для DDL триггеров тип триггера получается путём побитового ИЛИ над фазой события (0 - BEFORE, 1 - AFTER) и всех перечисленных типов событий: CREATE TABLE - 0x0000000000004002; ALTER TABLE - 0x0000000000004004; DROP TABLE - 0x0000000000004008; CREATE PROCEDURE - 0x0000000000004010 ALTER PROCEDURE - 0x0000000000004020; DROP PROCEDURE - 0x0000000000004040; CREATE FUNCTION - 0x0000000000004080; ALTER FUNCTION - 0x0000000000004100; DROP FUNCTION - 0x0000000000004200; CREATE TRIGGER - 0x0000000000004400; ALTER TRIGGER - 0x0000000000004800; DROP TRIGGER - 0x0000000000005000; CREATE EXCEPTION - 0x0000000000014000

(разрыв таблицы)

(разрыв таблицы)

Идентификатор столбца	Тип данных	Описание
RDB$TRIGGER_TYPE	BIGINT	ALTER EXCEPTION - 0x0000000000024000; DROP EXCEPTION - 0x0000000000044000; CREATE VIEW - 0x0000000000084000; ALTER VIEW - 0x0000000000104000; DROP VIEW - 0x0000000000204000; CREATE DOMAIN - 0x0000000000404000; ALTER DOMAIN - 0x0000000000804000; DROP DOMAIN - 0x0000000001004000; CREATE ROLE - 0x0000000002004000; ALTER ROLE - 0x0000000004004000; DROP ROLE - 0x0000000008004000; CREATE INDEX - 0x0000000010004000; ALTER INDEX - 0x0000000020004000; DROP INDEX - 0x0000000040004000; CREATE SEQUENCE - 0x0000000080004000; ALTER SEQUENCE - 0x0000000100004000; DROP SEQUENCE - 0x0000000200004000; CREATE USER - 0x0000000400004000; ALTER USER - 0x0000000800004000; DROP USER - 0x0000001000004000; CREATE COLLATION - 0x0000002000004000; DROP COLLATION - 0x0000004000004000; ALTER CHARACTER SET - 0x0000008000004000; CREATE PACKAGE - 0x0000010000004000; ALTER PACKAGE - 0x0000020000004000; DROP PACKAGE - 0x0000040000004000; CREATE PACKAGE BODY - 0x0000080000004000; DROP PACKAGE BODY - 0x0000100000004000; CREATE MAPPING - 0x0000200000004000; ALTER MAPPING - 0x0000400000004000; DROP MAPPING - 0x0000800000004000; ANY DDL STATEMENT - 0x7FFFFFFFFFFFDFFE.
RDB$TRIGGER_SOURCE	BLOB TEXT	Хранит исходный код триггера в PSQL.
RDB$TRIGGER_BLR	BLOB BLR	Хранит триггер в двоичном коде BLR.
RDB$DESCRIPTION	BLOB TEXT	Текст примечания триггера.
RDB$TRIGGER_INACTIVE	SMALLINT	Указывает, является ли триггер в настоящее время неактивным (1) или активным (0).
RDB$SYSTEM_FLAG	SMALLINT	Признак — триггер определен пользователем (0) или системой (1 или выше).
RDB$FLAGS	SMALLINT	Внутреннее использование.

(разрыв таблицы)

(разрыв таблицы)

Идентификатор столбца	Тип данных	Описание
RDB$VALID_BLR	SMALLINT	Указывает, остается ли текст триггера корректным после последнего изменения триггера при помощи оператора ALTER TRIGGER.
RDB$DEBUG_INFO	BLOB	Содержит отладочную информацию о переменных, используемых в триггере.
RDB$ENGINE_NAME	CHAR(63)	Имя движка для использования внешних триггеров. Обычно UDR.
RDB$ENTRYPOINT	CHAR(255)	Имя точки входа в библиотеке, где находится этот триггер.
RDB$SQL_SECURITY	BOOLEAN	Определяет в контексте какого пользователя будет выполняться триггер. Если установлен в FALSE, то триггер выполняется с правами вызвавшего его пользователя. Иначе триггер выполняется с правами его владельца (создателя)

RDB$TYPES

Описание перечислимых типов данных.

Идентификатор столбца	Тип данных	Описание
RDB$FIELD_NAME	CHAR(63)	Имя столбца, для которого определен данный перечислимый тип.
RDB$TYPE	SMALLINT	Задает идентификатор для типа. Последовательность чисел является уникальной для каждого отдельного перечислимого типа: 0 — таблица, 1 — представление, 2 — триггер, 3 — вычисляемый столбец, 4 — проверка, 5 — процедура.
RDB$TYPE_NAME	CHAR(63)	Текстовое представление для перечислимого типа.
RDB$DESCRIPTION	BLOB TEXT	Произвольный текст примечания к перечислимому типу.
RDB$SYSTEM_FLAG	SMALLINT	Признак: определен пользователем (значение 0) или системой (значение 1 или выше).

RDB$PUBLICATIONS

Хранит публикации репликации, заданные в базе данных.

Идентификатор столбца	Тип данных	Описание
RDB$PUBLICATION_NAME	CHAR(63)	Имя публикации.
RDB$OWNER_NAME	CHAR(63)	Имя пользователя, создавшего публикацию.
RDB$SYSTEM_FLAG	SMALLINT	Системный флаг: 0 - если публикация определена пользователем; 1 или выше - если публикация определена системой.
RDB$ACTIVE_FLAG	SMALLINT	Неактивная (0) или активная (1).
RDB$AUTO_ENABLE	SMALLINT	Автоматическое добавление новых таблиц в публикацию: 0 - отключено; 1 - включено (таблицы будут автоматически добавляться в публикацию).

RDB$PUBLICATION_TABLES

Хранит имена таблиц, которые реплицируются в рамках публикации.

Идентификатор столбца	Тип данных	Описание
RDB$PUBLICATION_NAME	CHAR(63)	Имя публикации.
RDB$TABLE_NAME	CHAR(63)	Имя таблицы.

RDB$USER_PRIVILEGES

Полномочия пользователей системы.

Идентификатор столбца	Тип данных	Описание
RDB$USER	CHAR(63)	Объект, которому предоставляется данное полномочие. Если в качестве грантополучателя используется системная привилегия, то вместо имени системной привилегии в данное поле попадает значение перечисляемого типа RDB$SYSTEM_PRIVILEGES: 1 – USER_MANAGEMENT; 2 – READ_RAW_PAGES; 3 – CREATE_USER_TYPES; 4 – USE_NBACKUP_UTILITY; 5 – CHANGE_SHUTDOWN_MODE; 6 – TRACE_ANY_ATTACHMENT;

(разрыв таблицы)

(разрыв таблицы)

Идентификатор столбца	Тип данных	Описание
RDB$USER	CHAR(63)	7 – MONITOR_ANY_ATTACHMENT; 9 – CREATE_DATABASE; 10 – DROP_DATABASE; 11 – USE_GBAK_UTILITY; 12 – USE_GSTAT_UTILITY; 13 – USE_GFIX_UTILITY; 14 – IGNORE_DB_TRIGGERS; 15 – CHANGE_HEADER_SETTINGS; 16 – SELECT_ANY_OBJECT_IN_DATABASE; 17 – ACCESS_ANY_OBJECT_IN_DATABASE; 18 – MODIFY_ANY_OBJECT_IN_DATABASE; 19 – CHANGE_MAPPING_RULES; 20 – USE_GRANTED_BY_CLAUSE; 21 – GRANT_REVOKE_ON_ANY_OBJECT; 22 – GRANT_REVOKE_ANY_DDL_RIGHT; 23 – CREATE_PRIVILEGED_ROLES.
RDB$GRANTOR	CHAR(63)	Имя пользователя, предоставляющего полномочие.
RDB$PRIVILEGE	CHAR(6)	Привилегия, предоставляемая в полномочии: A - all (все привилегии), S - select (выборка данных), I - insert (добавление данных), D - delete (удаление строк), R - reference (внешний ключ), U - update (изменение данных), X - execute (выполнение), G - usage (использование), C - create (создание), L - alter (изменение), O - drop (удаление), M - membership (членство).
RDB$GRANT_OPTION	SMALLINT	Содержит ли полномочие авторизацию WITH GRANT OPTION: 0 — не содержит, 1 — содержит, 2 — .
RDB$RELATION_NAME	CHAR(63)	Имя объекта (таблица или роль), которому предоставляется полномочие.
RDB$FIELD_NAME	CHAR(63)	Имя столбца, к которому применяется привилегия на уровне столбца (только привилегии UPDATE и REFERENCES). Если предоставляется членство в роли, то в данном столбце будет значение NULL. Если роль предоставляется с использованием ключевого слова DEFAULT, то в данном столбце будет значение D.

(разрыв таблицы)

(разрыв таблицы)

Идентификатор столбца	Тип данных	Описание
RDB$USER_TYPE	SMALLINT	Идентифицирует тип пользователя, которому предоставляется привилегия: 1 – представление; 2 – триггер; 5 – процедура; 8 – пользователь; 13 – роль; 15 – функция; 18 – пакет.
RDB$OBJECT_TYPE	SMALLINT	Идентифицирует тип объекта, к которому предоставляется привилегия: 0 - таблица, представление; 2 - триггер; 3 - вычисляемый столбец; 4 - проверочное ограничение; 5 - процедура; 6 - выражение, используемое для индекса; 7 - исключение; 8 - пользователь; 9 - столбец; 10 - индекс; 11 - набор символов; 12 - группа пользователя; 13 - роль; 14 - генератор; 15 - UDF; 16 - BLOB фильтр; 17 - порядок сортировки; 18 - заголовок пакета; 19 - тело пакета; 20 - привилегия; Идентифицирует тип объекта, к которому предоставляется DDL-привилегия: 21 - база данных; 22 - таблица; 23 - представление; 24 - процедура; 25 - фунцкия; 26 - пакет; 27 - генератор; 28 - домен; 29 - исключение; 30 - роль; 31 - набор символов; 32 - порядок сортировки; 33 - фильтр; 34 - задания; 35,36 - табличные пространства; 37 - условие частичного индекса.

RDB$VIEW_RELATIONS

Описывает представления. Не используется в настоящей версии.

Идентификатор столбца	Тип данных	Описание
RDB$VIEW_NAME	CHAR(63)	Имя представления.
RDB$RELATION_NAME	CHAR(63)	Имя таблицы, на которое ссылается данное представление.
RDB$VIEW_CONTEXT	SMALLINT	Псевдоним, используемый для ссылки на столбец представления. Имеет то же значение, что и псевдоним, используемый в самом тексте представления на языке BLR в операторе запроса этого представления.
RDB$CONTEXT_NAME	CHAR(255)	Текстовый вариант псевдонима, указанного в столбце RDB$VIEW_CONTEXT.
RDB$CONTEXT_TYPE	SMALLINT	Тип контекста: 0 – таблица; 1 – представление; 2 – хранимая процедура
RDB$PACKAGE_NAME	CHAR(63)	Имя пакета для упакованной хранимой процедуры.

Поле RDB$VALID_BLR

В системных таблицах RDB$PROCEDURES, RDB$FUNCTIONS и RDB$TRIGGERS присутствует поле RDB$VALID_BLR. Оно предназначено для сигнализации о возможной недействительности модуля PSQL (процедуры или триггера) после изменения доменов или столбцов таблиц, от которых он зависит. При возникновении описанной выше ситуации поле RDB$VALID_BLR устанавливается в 0 для процедур или триггеров, код которых возможно является недействительным.

В триггерах, процедурах и функциях (в том числе и в процедурах и функциях пакетов) зависимости возникают от столбцов таблицы к которой они обращаются, а так же от любого параметра или переменной которые определены в модуле с использованием предложения TYPE OF.

После того, как ядро РЕД Базы Данных изменило любой домен, включая неявные домены, создаваемые внутри при определении столбцов или параметров, РЕД База Данных производит внутреннюю перекомпиляцию всех зависимостей.

Любой модуль, который не удалось перекомпилировать из-за несовместимости, возникающей из-за изменения домена, помечается как недействительный (поле RDB$VALID_BLR устанавливается в 0 в записи соответствующей системной таблице RDB$PROCEDURES или RDB$TRIGGERS).

Возобновление (установка RDB$VALID_BLR в 1) происходит когда:

1. домен изменён снова и его новое определение совместимо с ранее недействительным определением модуля; или
2. ранее недействительный модуль изменён так, что соответствовать новому определению домена.

Нижеприведённый запрос находит процедуры и триггеры, зависящие от определённого домена (в примере это домен 'MYDOMAIN'), и выводит информацию о состоянии поля RDB$VALID_BLR. Замените MYDOMAIN фактическим именем проверяемого домена. Используйте заглавные буквы, если домен создавался нечувствительным к регистру — в противном случае используйте точное написание имени домена с учётом регистра.

```
WITH VALID_PSQL (
   PSQL_TYPE,
   ROUTE_NAME,
   VALID)
AS ( SELECT 'Procedure', RDB$PROCEDURE_NAME, RDB$VALID_BLR
     FROM RDB$PROCEDURES
     WHERE RDB$PROCEDURES.RDB$PACKAGE_NAME IS NULL
     UNION ALL
        SELECT 'Function', RDB$FUNCTION_NAME, RDB$VALID_BLR
        FROM RDB$FUNCTIONS
        WHERE RDB$FUNCTIONS.RDB$PACKAGE_NAME IS NULL
     UNION ALL
        SELECT 'Package', RDB$PACKAGE_NAME, RDB$VALID_BODY_FLAG
        FROM RDB$PACKAGES
     UNION ALL
        SELECT 'Trigger', RDB$TRIGGER_NAME, RDB$VALID_BLR
        FROM RDB$TRIGGERS
        WHERE RDB$TRIGGERS.RDB$SYSTEM_FLAG = 0)
SELECT
   PSQL_TYPE,
   ROUTE_NAME,
   VALID
FROM VALID_PSQL
WHERE
   EXISTS ( SELECT * FROM RDB$DEPENDENCIES
            WHERE RDB$DEPENDENT_NAME = VALID_PSQL.ROUTE_NAME
            AND RDB$DEPENDED_ON_NAME = 'MYDOMAIN' );
```

Следующий запрос находит процедуры и триггеры, зависящие от определённого столбца таблицы (в примере это столбец `MYCOLUMN` таблицы `MYTABLE`), и выводит информацию о состоянии поля `RDB$VALID_BLR`. Замените `MYTABLE` и `MYCOLUMN` фактическими именами проверяемой таблицы и её столбца. Используйте заглавные буквы, если таблица и её столбец создавались нечувствительными к регистру — в противном случае используйте точное написание имени таблицы и её столбца с учётом регистра.

```
WITH VALID_PSQL (
   PSQL_TYPE,
   ROUTE_NAME,
   VALID)
AS ( SELECT 'Procedure', RDB$PROCEDURE_NAME, RDB$VALID_BLR
     FROM RDB$PROCEDURES
     WHERE RDB$PROCEDURES.RDB$PACKAGE_NAME IS NULL
     UNION ALL
        SELECT 'Function', RDB$FUNCTION_NAME, RDB$VALID_BLR
        FROM RDB$FUNCTIONS
        WHERE RDB$FUNCTIONS.RDB$PACKAGE_NAME IS NULL
     UNION ALL
        SELECT 'Package', RDB$PACKAGE_NAME, RDB$VALID_BODY_FLAG
        FROM RDB$PACKAGES
     UNION ALL
        SELECT 'Trigger', RDB$TRIGGER_NAME, RDB$VALID_BLR
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
        FROM RDB$TRIGGERS
        WHERE RDB$TRIGGERS.RDB$SYSTEM_FLAG = 0)
SELECT
   PSQL_TYPE,
   ROUTE_NAME,
   VALID
FROM VALID_PSQL
WHERE
   EXISTS ( SELECT *
            FROM RDB$DEPENDENCIES D
            WHERE D.RDB$DEPENDENT_NAME = VALID_PSQL.ROUTE_NAME
            AND D.RDB$DEPENDED_ON_NAME = 'MYTABLE'
            AND D.RDB$FIELD_NAME = 'MYCOLUMN' );
```

Все случаи возникновения недействительных модулей, вызванных изменениями доменов/столбцов, отражаются в поле `RDB$VALID_BLR`. Тем не менее, другие виды изменения, таких как изменения количества входных или выходных параметров процедур и так далее, не влияют на поле проверки, даже если потенциально они могут привести к недействительности модуля. Типичные сценарии могут быть следующими:

1. Процедура (B) определена так, что она вызывает другую процедуру (A) и считывает выходные параметры из неё. В этом случае зависимость будет зарегистрирована в `RDB$DEPENDENCIES`. В последствии вызываемая процедура (A) может быть изменена для изменения или удаления одного и более выходных параметров. Оператор `ALTER PROCEDURE A` приведёт к ошибки при выполнении фиксации транзакции.
2. Процедура (B) вызывает процедуру (A), передавая ей значения в качестве входных параметров. Никаких зависимостей не будет зарегистрировано в `RDB$DEPENDENCIES`. Последующие модификации входных параметров процедуры A будут позволены. Отказ произойдет во время выполнения, когда B вызовет A с несогласованным набором входных параметров.

Для модулей PSQL, наследованных от более ранних версий РЕД Базы Данных, поле `RDB$VALID_BLR` имеет значение `NULL`. Это не означает, что их BLR является недействительным.

Команды утилиты командной строки `isql SHOW PROCEDURES, SHOW FUNCTIONS` и `SHOW TRIGGERS` при выводе информации отмечают звёздочкой модули, у которых поле `RDB$VALID_BLR` равно 0. Команды `SHOW PROCEDURE <имя процедуры>, SHOW FUNCTION <имя функции>` и `SHOW TRIGGER <имя триггера>`, выводящие на экран код PSQL модуля, не сигнализируют пользователя о недопустимом BLR.

Приложение Г Системные пакеты

Г.1 RDB$TIME_ZONE_UTIL

Пакет RDB$TIME_ZONE_UTIL содержит процедуру TRANSITIONS и функцию DATABASE_VERSION для работы с часовыми поясами.

Г.1.1 Функция DATABASE_VERSION

Функция RDB$TIME_ZONE_UTIL.DATABASE_VERSION возвращает версию базы данных часовых поясов (из библиотеки icu) типа VARCHAR(10) CHARACTER SET ASCII.

```
SELECT rdb$time_zone_util.database_version()
FROM rdb$database;

DATABASE_VERSION
===============
2024a
```

Г.1.2 Процедура TRANSITIONS

Процедура RDB$TIME_ZONE_UTIL.TRANSITIONS возвращает набор правил для часового пояса между начальной и конечной временной меткой.

Таблица Г.1 — Входные параметры процедуры TRANSITIONS

Параметр	Тип	Описание
RDB$TIME_ZONE_NAME	CHAR(63)	Наименование часового пояса
RDB$FROM_TIMESTAMP	TIMESTAMP WITH TIME ZONE	Начало интервала дат
RDB$TO_TIMESTAMP	TIMESTAMP WITH TIME ZONE	Окончание интервала дат

Таблица Г.2 — Выходные параметры процедуры TRANSITIONS

Параметр	Тип	Описание
RDB$START_TIMESTAMP	TIMESTAMP WITH TIME ZONE	Дата начала действия правила
RDB$END_TIMESTAMP	TIMESTAMP WITH TIME ZONE	Дата окончания действия правила
RDB$ZONE_OFFSET	SMALLINT	Смещение времени в минутах для заданного часового пояса
RDB$DST_OFFSET	SMALLINT	Летнее смещение времени в минутах для заданного часового пояса
RDB$EFFECTIVE_OFFSET	SMALLINT	Эффективное смещение, вычисляется как RDB$ZONE_OFFSET + RDB$DST_OFFSET

```
SELECT
   RDB$START_TIMESTAMP AS START_TIMESTAMP,
   RDB$END_TIMESTAMP AS END_TIMESTAMP,
   RDB$ZONE_OFFSET AS ZONE_OFF,
   RDB$DST_OFFSET AS DST_OFF,
   RDB$EFFECTIVE_OFFSET AS OFF
FROM rdb$time_zone_util.transitions('America/Sao_Paulo' ,
                                    timestamp '2017-01-01' ,
                                    timestamp '2019-01-01' );

START_TIMESTAMP               END_TIMESTAMP                 ZONE_OFF  DST_OFF  OFF
============================  ============================  ========  =======  ====
2016-10-16 03:00:00.0000 GMT  2017-02-19 01:59:59.9999 GMT  -180      60       -120
2017-02-19 02:00:00.0000 GMT  2017-10-15 02:59:59.9999 GMT  -180      0        -180
2017-10-15 03:00:00.0000 GMT  2018-02-18 01:59:59.9999 GMT  -180      60       -120
2018-02-18 02:00:00.0000 GMT  2018-11-04 02:59:59.9999 GMT  -180      0        -180
2018-11-04 03:00:00.0000 GMT  2019-02-17 01:59:59.9999 GMT  -180      60       -120
```

Г.2 RDB$BLOB_UTIL

Пакет RDB$BLOB_UTIL предназначен для обработки данных BLOB напрямую, чего не могут делать стандартные функции, такие как BLOB_APPEND и SUBSTRING.

Г.2.1 Функция NEW_BLOB

Функция NEW_BLOB используется для создания нового BLOB. Она возвращает BLOB, предназначенный для добавления данных, как BLOB_APPEND. Преимущество перед BLOB_APPEND заключается в том, что можно задать опции SEGMENTED и TEMP_STORAGE. BLOB_APPEND всегда создает BLOB во временном хранилище, что может быть не лучшим подходом, если созданный BLOB будет храниться в постоянной таблице, поскольку потребуется копирование. BLOB, возвращаемый функцией NEW_BLOB, даже если TEMP_STORAGE = FALSE, может использоваться с BLOB_APPEND для добавления данных.

Таблица Г.3 — Входные параметры функции NEW_BLOB

Параметр	Тип	Описание
SEGMENTED	BOOLEAN NOT NULL	Тип BLOB: true - сегментированный, false - потоковый
TEMP_STORAGE	BOOLEAN NOT NULL	Место хранения BLOB: true - во временном пространстве, false - в базе данных

Пример создания BLOB во временном пространстве и возвращение его в EXECUTE BLOCK:

```
execute block returns (b blob)
as
begin
   -- Создание дескриптора BLOB во временном пространстве.
   b = rdb$blob_util.new_blob(false, true);

   -- Добавление фрагментов данных.
   b = blob_append(b, '12345');
   b = blob_append(b, '67');
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
    suspend;
end!
```

Г.2.2 Функция OPEN_BLOB

Функция OPEN_BLOB используется для открытия существующего BLOB для чтения. Она возвращает дескриптор (целое число, связанное с транзакцией), подходящий для использования с другими функциями этого пакета, такими как SEEK, READ_DATA и CLOSE_HANDLE.

Таблица Г.4 — Входные параметры функции OPEN_BLOB

Параметр	Тип	Описание
BLOB	BLOB NOT NULL	Переменная типа BLOB

Г.2.3 Функция IS_WRITABLE

Функция IS_WRITABLE возвращает true, если BLOB подходит для добавления данных без копирования с помощью BLOB_APPEND.

Таблица Г.5 — Входные параметры функции IS_WRITABLE

Параметр	Тип	Описание
BLOB	BLOB NOT NULL	Переменная типа BLOB

Пример проверки BLOB на доступность для записи:

```
create table t(b blob);

set term !;

execute block returns (bool boolean)
as
  declare b blob;
begin
  b = blob_append(null, 'writable');
  bool = rdb$blob_util.is_writable(b);
  suspend;

  insert into t (b) values ('not writable') returning b into b;
  bool = rdb$blob_util.is_writable(b);
  suspend;
end!

set term ;!
```

Г.2.4 Функция READ_DATA

Функция READ_DATA используется для чтения фрагментов данных по дескриптору BLOB, открытому с помощью RDB$BLOB_UTIL.OPEN_BLOB. Возвращает NULL, когда BLOB полностью прочитан. Если LENGTH передается с положительным числом, он возвращает VARBINARY с его максимальной длиной. Если LENGTH равна NULL, то возвращается только сегмент BLOB с максимальной длиной 32765.

Таблица Г.6 — Входные параметры функции READ_DATA

Параметр	Тип	Описание
HANDLE	INTEGER NOT NULL	Дескриптор
LENGTH	INTEGER	Длина

Пример открытия BLOB и возвращения его частей с помощью EXECUTE BLOCK:

```
execute block returns (s varchar(10))
as
   declare b blob = '1234567';
   declare bhandle integer;
begin
   -- Открытие BLOB и получение дескриптора BLOB.
   bhandle = rdb$blob_util.open_blob(b);

   -- Получение и возвращение фрагментов данных в виде строки.
   s = rdb$blob_util.read_data(bhandle, 3);
   suspend;

   s = rdb$blob_util.read_data(bhandle, 3);
   suspend;

   s = rdb$blob_util.read_data(bhandle, 3);
   suspend;

   -- Здесь BLOB полностью прочитан, поэтому возвращается NULL.
   s = rdb$blob_util.read_data(bhandle, 3);
   suspend;

   -- Закрытие дескриптора BLOB.
   execute procedure rdb$blob_util.close_handle(bhandle);
end!
```

Г.2.5 Функция SEEK

Функция SEEK возвращает новую позицию для следующего READ_DATA. MODE может быть 0 (с начала), 1 (с текущей позиции) или 2 (с конца). Если MODE равен 2, OFFSET должен быть нулевым или отрицательным.

Таблица Г.7 — Входные параметры функции SEEK

Параметр	Тип	Описание
HANDLE	INTEGER NOT NULL	Дескриптор
MODE	INTEGER NOT NULL	Режим
OFFSET	INTEGER NOT NULL	Смещение

Пример работы функции SEEK:

```
set term !;
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
execute block returns (s varchar(10))
as
   declare b blob;
   declare bhandle integer;
begin
   -- Создание дескриптора потокового BLOB
   b = rdb$blob_util.new_blob(false, true);

   -- Добавление данных.;
   b = blob_append(b, '0123456789');

   -- Открытие BLOB.;
   bhandle = rdb$blob_util.open_blob(b);

   -- Поиск с начала до 5.;

   rdb$blob_util.seek(bhandle, 0, 5);
   s = rdb$blob_util.read_data(bhandle, 3);
   suspend;

   -- Поиск с начала до 2.
   rdb$blob_util.seek(bhandle, 0, 2);
   s = rdb$blob_util.read_data(bhandle, 3);
   suspend;

  -- Ещё на 2 вперед.
  rdb$blob_util.seek(bhandle, 1, 2);
  s = rdb$blob_util.read_data(bhandle, 3);
  suspend;

  -- Поиск с конца на один элемент назад.
  rdb$blob_util.seek(bhandle, 2, -1);
  s = rdb$blob_util.read_data(bhandle, 3);
  suspend;
end!
set term ;!
```

Г.2.6 Процедура CANCEL_BLOB

Процедура `CANCEL_BLOB` используется для немедленного освобождения временного `BLOB`, например, созданного с помощью `BLOB_APPEND`.

Обратите внимание, что если тот же `BLOB` будет использован после отмены, используя ту же переменную или другую переменную с той же ссылкой на `BLOB id`, будет выдана ошибка `invalid blob id`.

Г.2.7 Процедура CLOSE_HANDLE

Процедура `CLOSE_HANDLE` используется для закрытия дескриптора `BLOB`, открытого с помощью `OPEN_BLOB`. Незакрытые дескрипторы закрываются автоматически только в конце транзакции.

Таблица Г.8 — Входные параметры процедуры `CLOSE_HANDLE`

Параметр	Тип	Описание
`HANDLE`	`INTEGER NOT NULL`	Дескриптор

Приложение Д Наборы символов и порядки сортировки

Наборы символов в РЕД База Данных и соответствующие им порядки сортировки представлены в таблице Д.1.

Таблица Д.1 — Наборы символов и порядки сортировки РЕД База Данных

ID	Название	Байт/ символ	Порядок сортировки	Язык
2	ASCII	1	ASCII	Английский
56	BIG_5	2	BIG_5	Китайский Вьетнамский Корейский
68	CP943C	2	CP943C CP943C_UNICODE	Японский Японский
50	CYRL	1	CYRL DB_RUS PDOX_CYRL	Русский Русский dBase Русский Paradox
10	DOS437	1	DOS437 DB_DEU437 DB_ESP437 DB_FIN437 DB_FRA437 DB_ITA437 DB_NLD437 DB_SVE437 DB_UK437 DB_US437 PDOX_ASCII PDOX_SWEDFIN PDOX_INTL	Английский—США Немецкий dBase Испанский dBase Финский dBase Французский dBase Итальянский dBase Голландский dBase Шведский dBase Английский (Великобритания) dBase Английский (США) dBase Кодовая страница Paradox — ASCII Шведский / Финский Paradox Международный английский Paradox
9	DOS737	1	DOS737	Греческий
15	DOS775	1	DOS775	Балтийский
11	DOS850	1	DOS850 DB_DEU850 DB_ESP850 DB_FRA850 DB_FRC850 DB_ITA850 DB_NLD850 DB_PTB850 DB_SVE850 DB_UK850 DB_US850	Латинский I (нет символа Евро) Немецкий Испанский Французский Французский — Канада Итальянский Голландский Португальский — Бразилия Шведский Английский — Великобритания Английский — США

(разрыв таблицы)

(разрыв таблицы)

ID	Название	Байт/ символ	Порядок сортировки	Язык
45	DOS852	1	DOS852 DB_CSY DB_PLK DB_SLO PDOX_CSY PDOX_HUN PDOX_PLK PDOX_SLO	Латинский II Чешский dBase Польский dBase Словацкий dBase Чешский Paradox Венгерский Paradox Польский Paradox Словацкий Paradox
46	DOS857	1	DOS857 DB_TRK	Турецкий Турецкий dBase
16	DOS858	1	DOS858	Латинский I с символом Евро
13	DOS860	1	DOS860 DB_PTG860	Португальский Португальский dBase
47	DOS861	1	DOS861 PDOX_ISL	Исландский Исландский Paradox
17	DOS862	1	DOS862	Иврит
14	DOS863	1	DOS863 DB_FRC863	Французский — Канада Французский dBase — Канада
18	DOS864	1	DOS864	Арабский
12	DOS865	1	DOS865 DB_DAN865 DB_NOR865 PDOX_NORDAN4	Скандинавские Датский dBase Норвежский dBase Paradox Норвегия и Дания.
48	DOS866	1	DOS866	Русский
49	DOS869	1	DOS869	Современный греческий
6	EUCJ_0208	2	EUCJ_0208	Японские EUC
57	GB_2312	2	GB_2312	Упрощенный китайский (Гонконг, Корея)
69	GB18030	4	GB18030 GB18030_UNICODE	Китайский Китайский
67	GBK	2	GBK GBK_UNICODE	Китайский Китайский
21	ISO8859_1	1	ISO8859_1 DA_DA DE_DE DU_NL EN_UK EN_US ES_ES ES_ES_CI_AI FI_FI FR_CA FR_FR	Латинский I Датский Немецкий Голландский Английский, Великобритания Английский, США Испанский Испанский не чувствительный к регистру символов и к акцентам (ударению) Финский Французский, Канада Французский

(разрыв таблицы)

(разрыв таблицы)

ID	Название	Байт/ символ	Порядок сортировки	Язык
21	ISO8859_1	1	FR_FR_CI_AI	Французский — не чувствительный к регистру символов и к акцентам (ударению)
			IS_IS	Исландский
			IT_IT	Итальянский
			NO_NO	Норвежский
			PT_PT	Португальский
			PT_BR	Португальский, Бразилия
			SV_SV	Шведский
22	ISO8859_2	1	ISO8859_2	Латинский 2 — Центральная Европа (хорватский, чешский, венгерский, польский, румынский, сербский, словацкий, словенский)
			CS_CZ	Чешский
			ISO_HUN	Венгерский — не чувствительный к регистру и чувствительный к акценту (ударению)
			ISO_PLK	Польский.
23	ISO8859_3	1	ISO8859_3	Латинский 3 — Южная Европа (мальтийский, эсперанто).
34	ISO8859_4	1	ISO8859_4	Латинский 4 — Северная Европа (эстонский, латвийский, литовский, гренландский, саамский).
35	ISO8859_5	1	ISO8859_5	Кириллица (русский).
36	ISO8859_6	1	ISO8859_6	Арабский
37	ISO8859_7	1	ISO8859_7	Греческий
38	ISO8859_8	1	ISO8859_8	Иврит
39	ISO8859_9	1	ISO8859_9	Латинский 5
40	ISO8859_13	1	ISO8859_13	Латинский 7 — Балтика
			LT_LT	Литовский
63	KOI8R	1	KOI8R	Русский - словарное упорядочение
			KOI8R_RU	Русский
64	KOI8U	1	KOI8U	Украинский - словарное упорядочение
			KOI8R_UA	Украинский
44	KSC_5601	2	KSC_5601	Корейский
			KSC_DICTIONARY	Корейский — словарный порядок сортировки
19	NEXT	1	NEXT	Кодирование NeXTSTEP
			NXT_DEU	Немецкий
			NXT_ESP	Испанский
			NXT_FRA	Французский
			NXT_ITA	Итальянский
			NXT_US	Английский, США
0	NONE	1	NONE	Нейтральная кодовая страница. Перевод в верхний регистр выполняется только для кодов ASCII 97–122. Постарайтесь сделать так, чтобы этот набор символов никогда не появлялся в столбцах ваших баз данных.
1	OCTETS	1	OCTETS	Двоичные символы

(разрыв таблицы)

(разрыв таблицы)

ID	Название	Байт/ символ	Порядок сортировки	Язык
5	SJIS_0208	2	SJIS_0208	Японский
66	TIS620	1	TIS620 TIS620_UNICODE	Тайский Тайский
3	UNICODE_FSS	3	UNICODE_FSS	UNICODE
4	UTF8	4	UTF8 USC_BASIC UNICODE UNICODE_CI UNICODE_CI_AI	UNICODE 4.0 UNICODE 4.0 UNICODE 4.0 UNICODE 4.0. Не чувствительна к регистру символов UNICODE 4.0. Не чувствительна к регистру символов и к акцентам (ударению)
51	WIN1250	1	WIN1250 BS_BA PXW_CSY PXW_HUN PXW_HUNDC PXW_PLK PXW_SLOV WIN_CZ WIN_CZ_CI WIN_CZ_CI_AI	ANSI— Центральная Европа Боснийский Чешский Венгерский — не чувствительный к регистру и чувствительный к акценту (ударению) Венгерский — словарная сортировка Польский Словенский Чешский Чешский без различия строчных и прописных букв Чешский без различия строчных и прописных букв, нечувствительный к знакам ударения
52	WIN1251	1	WIN1251 WIN1251_UA PXW_CYRL	ANSI кириллица Украинский Paradox кириллица (русский)
53	WIN1252	1	WIN1252 PXW_INTL PXW_INTL850 PXW_NORDAN4 PXW_SPAN PXW_SWEDFIN WIN_PTBR	ANSI — Латинский I Английский интернациональный Paradox многоязыковый Латинский I Норвежский и датский Paradox испанский Шведский и финский Португальский, Бразилия
54	WIN1253	1	WIN1253 PXW_GREEK	ANSI греческий Paradox греческий
55	WIN1254	1	WIN1254 PXW_TURK	ANSI турецкий Paradox турецкий
58	WIN1255	1	WIN1255	ANSI иврит
59	WIN1256	1	WIN1256	ANSI арабский
60	WIN1257	1	WIN1257 WIN1257_EE WIN1257_LT WIN1257_LV	ANSI балтийский Эстонский. Словарное упорядочение Литовский. Словарное упорядочение Латвийский. Словарное упорядочение
65	WIN1258	1	WIN1258	Вьетнамский

Приложение E Функции, определенные пользователем (UDF)

Функции, определенные пользователем (User Defined Functions, UDF), — это программы, написанные на любом языке программирования, и хранящиеся в библиотеках `dll` (для Linux — `so`). Они существенно расширяют возможности SQL по обработке данных.

> Поддержка внешних UDF функций (реализованных в динамических библиотеках и определенных в базе данных с помощью функций оператора `DECLARE EXTERNAL FUNCTION`) в РЕД Базе Данных 5 устарела.

E.1 Объявление в базе данных UDF

Для того чтобы функция стала доступной в операторах SQL, необходимо выполнить оператор `DECLARE EXTERNAL FUNCTION`, который объявляет существующую функцию, определенную пользователем. Его синтаксис:

Листинг E.1. Синтаксис оператора `DECLARE EXTERNAL FUNCTION`

```
DECLARE EXTERNAL FUNCTION <имя UDF>
    [<тип данных> [{BY DESCRIPTOR} | NULL] | CSTRING (<целое>) [NULL]
    [, <тип данных> [{BY DESCRIPTOR} | NULL] | CSTRING (<целое>) [NULL] ...]]
RETURNS {
      <тип данных> [BY VALUE | BY DESCRIPTOR]
    | CSTRING (<целое>)
    | PARAMETER <номер> }
[FREE_IT]
ENTRY_POINT '<имя точки входа>'
MODULE_NAME '<имя модуля>';
```

Объявить внешнюю функцию может пользователь с административными привилегиями и пользователь с привилегией `CREATE FUNCTION`. Пользователь, объявивший внешнюю функцию, становится её владельцем.

Имя UDF — то имя, которое будет использоваться при обращении к функции в операторах SQL. Это имя может отличаться от имени точки входа.

После имени функции перечисляются входные параметры, передаваемые функции. Параметры разделяются запятыми. Для каждого параметра указывается либо тип данных SQL, либо ключевое слово `CSTRING`. Это ключевое слово означает, что параметр является строкой символов, которая заканчивается нулевым значением. В скобках за этим словом задается максимальное число символов, которое может присутствовать в строке, включая завершающее нулевое значение.

По умолчанию входные параметры передаются по ссылке. При передаче `NULL` значения по ссылке оно преобразовывается в эквивалент нуля, например, число 0 или пустую строку. Если после указанного параметра указано ключевое слово `NULL`, то при передаче значение `NULL` оно попадёт в функцию в виде нулевого указателя.

Обязательное предложение `RETURNS` описывает возвращаемый функцией выходной параметр. Функция UDF всегда возвращает ровно одно значение. Можно указать тип данных SQL, строку, завершающуюся нулевым значением (`CSTRING`) или ключевое слово `PARAMETER`, за которым следует число. Ключевое слово `PARAMETER` используется, когда возвращается значение типа `BLOB`. Номер задает порядковый номер входного возвращаемого параметра.

Функция может получать произвольное количество входных параметров или ни одного. Каждая

функция возвращает одно значение указанного типа. При обращении к функции, определенной пользователем, необходимо после имени функции в скобках указать передаваемые функции параметры. Если функция не получает входных параметров, то обязательно нужно записать круглые скобки.

Ключевое слово `BY VALUE` означает, что результат возвращается по значению, а не по ссылке (по умолчанию значение возвращается по ссылке).

Ключевое слово `BY DESCRIPTOR` задает передачу параметров по дескриптору.

Ключевое слово `FREE_IT` означает, что память, выделенная для хранения возвращаемого значения, должна быть освобождена после завершения выполнения функции. Применяется только в том случае, если эта память в UDF выделялась динамически.

Предложение `ENTRY_POINT` указывает имя точки входа для функции в модуле.

Предложение `MODULE_NAME` задает имя модуля, в котором находится описываемая функция. В ссылке на модуль может отсутствовать полный путь и расширение файла. По умолчанию динамические библиотеки пользовательских функций должны располагаться в папке UDF корневого каталога сервера. Параметр `UDFAccess` в файле `firebird.conf` позволяет изменить ограничения доступа к библиотекам внешних функций.

E.2 Изменение точки входа или имени модуля для UDF

Оператор `ALTER EXTERNAL FUNCTION` позволяет изменить в функции UDF имя точки входа и/или имя модуля. Его синтаксис:

Листинг E.2. Синтаксис оператора `ALTER EXTERNAL FUNCTION`

```
ALTER EXTERNAL FUNCTION <имя UDF>
[ENTRY_POINT '<имя точки входа>']
[MODULE_NAME '<имя модуля>'];
```

Имя UDF — имя существующей функции.

Предложение `ENTRY_POINT` указывает новое имя точки входа для функции в модуле.

Предложение `MODULE_NAME` задает новое имя модуля, в котором находится описываемая функция.

Изменить внешнюю функцию может администратор, владелец функции (ее создатель) или пользователь с привилегией `ALTER ANY FUNCTION`.

E.3 Удаление объявления UDF из базы данных

Оператор `DROP EXTERNAL FUNCTION` удаляет объявление функции определённой пользователем из базы данных. Если есть зависимости от внешней функции, то удаления не произойдёт и будет выдана соответствующая ошибка.

Листинг E.3. Синтаксис оператора `DROP EXTERNAL FUNCTION`

```
DROP EXTERNAL FUNCTION <имя UDF>;
```

Удалить внешнюю функцию может администратор, владелец функции (ее создатель) или пользователь с привилегией `DROP ANY FUNCTION`.

Приложение Ж Операторы SQL

В этом разделе описан синтаксис и краткое назначение операторов SQL. Операторы расположены в алфавитном порядке.

ALTER CHARACTER SET

С помощью оператора ALTER CHARACTER SET есть возможность изменять последовательность сортировки по умолчанию для указанного набора символов:

Листинг Ж.1. Синтаксис оператора ALTER CHARACTER SET

```
ALTER CHARACTER SET <набор символов>
SET DEFAULT COLLATION <сортировка>;
```

Это повлияет на использование набора символов в будущем, кроме случаев, когда явно переопределена сортировка COLLATE. Сортировка существующих доменов, столбцов и переменных PSQL при этом не будет изменена.

Если сортировка по умолчанию была изменена для набора символов базы данных, что был указан при создании базы данных, то также изменяется и сортировка для базы данных.

Если сортировка по умолчанию была изменена для набора символов, который был указан при подключении, строковые константы будут интерпретироваться в соответствии с новыми параметрами сортировки (если набор символов и/или сортировка не переопределяются).

```
ALTER CHARACTER SET win1252
SET DEFAULT COLLATION win_ptbr;
```

ALTER DATABASE

Оператор ALTER DATABASE/SHEMA позволяет изменить структуры файлов базы данных, переключить её в состояние «безопасное для копирования» или изменить некоторых свойств базы данных. Для выполнения оператора необходимо предварительно соединиться с базой данных (оператор CONNECT). Синтаксис оператора:

Листинг Ж.2. Синтаксис оператора ALTER DATABASE

```
ALTER {DATABASE | SCHEMA}
[ADD FILE <вторичный файл> [ADD FILE <вторичный файл> ...]]
[ADD DIFFERENCE FILE '<имя файла>' | DROP DIFFERENCE FILE]
[{BEGIN | END} BACKUP]
[SET DEFAULT CHARACTER SET <набор символов>]
[SET LINGER TO <секунды> | DROP LINGER]
[SET DEFAULT SQL SECURITY {DEFINER | INVOKER}]
[ENCRYPT WITH <плагин шифрования> [KEY <ключ шифрования>] | DECRYPT]

<вторичный файл> ::= FILE '<путь к файлу>'
                     [LENGTH [=] <целое> [PAGE[S]]]
                     [STARTING [AT [PAGE]] <целое>]
```

Изменять базу данных может ее владелец, администратор и пользователь с ролью ALTER DATABASE.

При использовании данного оператора можно:

- добавить к базе данных вторичный файл (ADD FILE);
- задать путь и имя дельта файла, в который будут записываться изменения, внесенные в базу данных после перевода ее в режим «безопасного копирования» (ADD DIFFERENCE FILE);
- удалить ранее указанное описание файла дельты с помощью оператора DROP DIFFERENCE FILE;
- переводить базу данных в режим «безопасного копирования» (BEGIN BACKUP) и возвращать в режим нормального функционирования (END BACKUP);
- изменять набор символов по умолчанию для базы данных (SET DEFAULT CHARACTER SET);
- установить задержку закрытия базы данных (SET LINGER);
- изменять поведение по умолчанию, которое определяет в контексте какого пользователя будет выполняться объект базы данных (процедура, функция, пакет, триггер, таблица) (SET DEFAULT SQL SECURITY);
- шифровать базу данных с помощью указанного плагина шифрования (ENCRYPT WITH) или дешифровать (DECRYPT)

Подробно оператор описан в главе 6.

Подробно оператор описан в главе Базы данных (DATABASE).

См. также операторы CREATE DATABASE, DROP DATABASE, CREATE SHADOW, DROP SHADOW, CONNECT.

ALTER DOMAIN

Оператор ALTER DOMAIN используется для изменения характеристик существующего домена. Синтаксис оператора:

Листинг Ж.3. Синтаксис оператора ALTER DOMAIN

```
ALTER DOMAIN <имя>
  [TO <новое имя>]
  [{ SET DEFAULT {<литерал> | NULL | <контекстная переменная>}
   | DROP DEFAULT}]
  [{ SET | DROP} NOT NULL]
  [{ ADD [CONSTRAINT] CHECK (<условие домена>)
   | DROP CONSTRAINT}]
  [TYPE <тип данных> [CHARACTER SET <набор символов> [COLLATE <порядок сортировки>]
]];
```

Изменить существующий домен может владелец домена (его создатель), пользователь с административными привилегиями или пользователь с привилегией ALTER ANY DOMAIN.

При использовании данного оператора можно:

- изменить имя домена (конструкция <имя> TO <новое имя>);
- установить новое значение по умолчанию (SET DEFAULT),;
- удалить существующее значение по умолчанию (DROP DEFAULT);
- задать новое условие домена (предложение ADD [CONSTRAINT] CHECK (<условие домена>));
- установить или удалить ограничение NOT NULL для домена;
- удалить существующее условие домена (предложение DROP CONSTRAINT);
- задать новый тип данных (TYPE <тип данных>);
- изменить набор символов (CHARACTER SET <набор символов>) и порядок сортировки (COLLATE <порядок сортировки>) для символьного домена.

Подробно оператор ALTER DOMAIN с примерами описан в главе 8.

См. также операторы CREATE DOMAIN, DROP DOMAIN.

ALTER EXCEPTION

Оператор позволяет изменить текст существующего в базе данных пользовательского исключения. Синтаксис:

Листинг Ж.4. Синтаксис оператора ALTER EXCEPTION

```
ALTER EXCEPTION <имя исключения> '<текст сообщения>';
```

Текст сообщения может содержать до 1021 символа.

Изменять текст сообщения пользовательского исключения может администратор, владелец исключения и пользователь с привилегией ALTER ANY EXCEPTION.

См. также операторы CREATE EXCEPTION, CREATE OR ALTER EXCEPTION, RECREATE EXCEPTION, DROP EXCEPTION, операторы PSQL WHEN-DO, EXCEPTION.

ALTER EXTERNAL CONNECTIONS POOL CLEAR ALL

Оператор позволяет закрыть все бездействующие соединения в пуле внешних соединений.

Листинг Ж.5. Синтаксис оператора ALTER EXTERNAL CONNECTIONS POOL CLEAR ALL

```
ALTER EXTERNAL CONNECTIONS POOL CLEAR ALL
```

Все активные соединения будут отсоединены от пула (такие соединения будут немедленно закрыты, когда они не будут использоваться).

Подробнее о пуле внешних соединений см. в «Руководстве администратора».

ALTER EXTERNAL CONNECTIONS POOL CLEAR OLDEST

Оператор позволяет закрыть бездействующие соединения в пуле внешних соединений, у которых истекло время жизни.

Листинг Ж.6. Синтаксис оператора ALTER EXTERNAL CONNECTIONS POOL CLEAR OLDEST

```
ALTER EXTERNAL CONNECTIONS POOL CLEAR OLDEST
```

Подробнее о пуле внешних соединений см. в «Руководстве администратора».

ALTER EXTERNAL CONNECTIONS POOL SET LIFETIME

Оператор позволяет устанавливать время жизни бездействующих соединений в пуле внешних соединений.

Листинг Ж.7. Синтаксис оператора ALTER EXTERNAL CONNECTIONS POOL SET LIFETIME

```
ALTER EXTERNAL CONNECTIONS POOL SET LIFETIME <значение> {SECOND | MINUTE | HOUR}
```

Допустимые значения от 1 секунды до 24 часов. Значение по умолчанию определяется в `firebird.conf` (параметр `ExtConnPoolLifeTime` в секундах).

Подробнее о пуле внешних соединений см. в «Руководстве администратора».

ALTER EXTERNAL CONNECTIONS POOL SET SIZE

Оператор позволяет устанавливать максимальное количество бездействующих соединений в пуле внешних соединений.

Листинг Ж.8. Синтаксис оператора ALTER EXTERNAL CONNECTIONS POOL SET SIZE

```
ALTER EXTERNAL CONNECTIONS POOL SET SIZE <размер>
```

Допустимые значения от 0 до 1000. Нулевое значение обозначает что пул выключен. Значение по умолчанию определяется в `firebird.conf` (параметр `ExtConnPoolSize`).

Подробнее о пуле внешних соединений см. в «Руководстве администратора».

ALTER EXTERNAL FUNCTION

Оператор позволяет изменить в функции, определенной пользователем (UDF— User Defined Function), имя точки входа и/или имя модуля. Его синтаксис:

Листинг Ж.9. Синтаксис оператора ALTER EXTERNAL FUNCTION

```
ALTER EXTERNAL FUNCTION <имя UDF>
[ENTRY_POINT '<имя точки входа>']
[MODULE_NAME '<имя модуля>'];
```

Изменить внешнюю функцию может администратор, владелец функции (ее создатель) или пользователь с привилегией `ALTER ANY FUNCTION`.

При использовании данного оператора можно:

- указать новое имя точки входа для функции в модуле (`ENTRY_POINT`);
- задать новое имя модуля, в котором находится описываемая функция (`MODULE_NAME`).

Подробно оператор описан в Приложение Е.

См. также оператор DECLARE EXTERNAL FUNCTION, DROP EXTERNAL FUNCTION.

ALTER FUNCTION

Для изменения существующей хранимой функции используется оператор `ALTER FUNCTION`. Синтаксис оператора:

Листинг Ж.10. Синтаксис оператора изменения хранимой функции ALTER FUNCTION

```
ALTER FUNCTION <имя хранимой функции>
   [(<входной параметр> [, <входной параметр> ...])]
RETURNS <тип> [COLLATE <сортировка>] [DETERMINISTIC]
[SQL SECURITY {DEFINER | INVOKER}]
{ EXTERNAL NAME '<внешний модуль>' ENGINE <имя движка> [AS <тело внешней функции>]}
|
{
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
    {AS '<BLR код>'}
    |
    {AS
       [<объявление> [<объявление> ...] ]
    BEGIN
       <блок операторов>
    END }
}

<входной параметр> ::= <описание параметра> [{=|DEFAULT} <значение по умолчанию>]

<описание параметра> ::= <имя параметра> <тип> [NOT NULL]
                         [COLLATE <порядок сортировки>]

<тип> ::= <тип данных SQL>
        | [TYPE OF] <имя домена>
        | TYPE OF COLUMN <имя таблицы/представления>.<имя столбца>

<значение по умолчанию> ::= {<литерал> | NULL | <контекстная переменная>}

<внешний модуль> ::= '<имя внешнего модуля>!<имя функции в модуле>[! <информация>]'

<объявление> ::= <объявление локальной переменной>;
               | <объявление курсора>;
               | <объявление процедуры/функции>
               | <реализация процедуры/функции>
```

Изменять хранимую функцию может администратор, владелец хранимой функции, пользователь с привилегией `ALTER ANY FUNCTION`.

Оператор `ALTER FUNCTION` позволяет изменять:

- состав и характеристики входных параметров;
- тип выходного значения;
- в контексте какого пользователя будет выполняться функция;
- локальные переменные, курсоры, подпрограммы;
- тело хранимой функции на языке SQL;
- исходный SQL код функции на двоичный BLR код;
- точку входа и имя движка (для внешних функций);
- а также указать, что функция является детерминированной.

После выполнения существующие привилегии и зависимости сохраняются.

Подробно оператор описан в главе 19.

См. также операторы CREATE OR ALTER FUNCTION, RECREATE FUNCTION, CREATE FUNCTION, DROP FUNCTION, DECLARE VARIABLE, DECLARE CURSOR, DECLARE FUNCTION, DECLARE PROCEDURE.

ALTER INDEX

Оператор ALTER INDEX позволяет сделать индекс активным или неактивным. Возможностей изменения структуры или упорядоченности его столбцов этот оператор не предусматривает. Его синтаксис:

Листинг Ж.11. Синтаксис оператора ALTER INDEX

```
ALTER INDEX <имя индекса>
{ { ACTIVE | INACTIVE }
| SET TABLESPACE [TO] {<имя табличного пространства> | PRIMARY}
}
```

Индекс может изменять только владелец таблицы, для которой создан индекс, администратор и пользователь с привилегией ALTER ANY TABLE.

Ключевое слово ACTIVE задает перевод неактивного индекса в активное состояние. Ключевое слово INACTIVE указывает, что индекс переводится в неактивное состояние. После перевода индекса из неактивного состояния в активное система заново создает весь индекс.

Для индекса может быть указано табличное пространство для отдельного физического хранения.

Подробно оператор описан в главе 14.

См. также операторы CREATE INDEX, DROP INDEX, SET STATISTICS.

ALTER JOB

Оператор ALTER JOB позволяет изменять задания для планировщика.

Листинг Ж.12. Синтаксис оператора ALTER JOB

```
ALTER JOB <имя_задания>
[<параметры расписания>]
[ACTIVE | INACTIVE]
[START DATE '<timestamp>' | NULL]
[END DATE '<timestamp>' | NULL]
[ AS
     <объявления переменных>
  BEGIN
     <блок sql-операторов>
  END
| COMMAND '<команда>' ]

<параметры расписания> ::= '<Минуты> <Часы> <Дни_месяца> <Месяцы> <Дни_недели>'
```

Изменить задание может его создатель и SYSDBA.

При использовании данного оператора можно изменить:

- его активность (будет ли задание запускаться или нет);
- расписание запуска;
- интервал времени, в котором запускается задание, или убирать его;
- команды ОС;
- локальные переменные, курсоры, подпрограммы;
- тело задания;

Подробнее о планировщике заданий можно узнать в главе 23.

См. также операторы CREATE JOB, DROP JOB.

ALTER MAPPING

Оператор ALTER MAPPING позволяет изменять любые опции существующего отображения.

Листинг Ж.13. Синтаксис оператора ALTER MAPPING

```
ALTER [GLOBAL] MAPPING <имя отображения>
USING {
   PLUGIN <имя плагина> [IN <имя базы данных>]
 | ANY PLUGIN [IN <имя базы данных> | SERVERWIDE]
 | MAPPING [IN <имя базы данных>]
 | '*' [IN <имя базы данных>] }
FROM { ANY <тип отображаемого объекта> | <тип отображаемого объекта> <имя
отображаемого объекта> }
TO { USER | ROLE } [<имя объекта, на которое произведено отображение>]
```

Изменить отображение может SYSDBA, владелец базы данных (если отображение локальное), пользователь с ролью RDB$ADMIN или пользователь root (Linux).

При использовании данного оператора можно:

- изменить источник отображения;
- изменить отображаемый объект;
- изменить пользователя или роль, на которого будет произведено отображение.

Подробнее синтаксис оператора разобран в главе 26.

См. также операторы CREATE OR ALTER MAPPING, DROP MAPPING, CREATE MAPPING.

ALTER PACKAGE

Оператор ALTER PACKAGE изменяет заголовок пакета. Синтаксис оператора:

Листинг Ж.14. Синтаксис оператора изменения заголовка пакета ALTER PACKAGE

```
ALTER PACKAGE <имя пакета>
[SQL SECURITY {DEFINER | INVOKER}]
AS
BEGIN
   [ <объявление процедуры> | <объявление функции> ...]
END

<объявление процедуры> ::=
   PROCEDURE <имя процедуры> [(<входной параметр> [, <входной параметр> ...])]
      [RETURNS (<выходной параметр> [, <выходной параметр> ...])]

<объявление функции> ::=
   FUNCTION <имя функции> [(<входной параметр> [, <входной параметр> ...])]
      RETURNS <тип> [COLLATE <сортировка>] [DETERMINISTIC]

<входной параметр> ::= <описание параметра> [{=|DEFAULT} <значение по умолчанию>]

<выходной параметр> ::= <описание параметра>

<описание параметра> ::= <имя параметра> <тип> [NOT NULL]
                         [COLLATE <порядок сортировки>]
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
<тип> ::= {
   <тип данных SQL>
 | [TYPE OF] <имя домена>
 | TYPE OF COLUMN <имя таблицы/представления>.<имя столбца> }

<значение по умолчанию>::= {<литерал> | NULL | <контекстная переменная>}
```

Изменить заголовок пакета может администратор, владелец пакета или пользователь с привилегией `ALTER ANY PACKAGE`.

При использовании данного оператора позволяется изменять:

- количество и состав процедур и функций;
- входные и выходные параметры входящих в пакет процедур и функций;
- в контексте какого пользователя будет выполняться пакет.

Состояние соответствия тела пакета его заголовку отображается в таблице `RDB$PACKAGES` в столбце `RDB$VALID_BODY_FLAG`.

Подробнее о пакетах описано в главе 21.

См. также операторы RECREATE PACKAGE, DROP PACKAGE, CREATE PACKAGE, CREATE OR ALTER PACKAGE, CREATE PROCEDURE, CREATE FUNCTION.

ALTER POLICY

Оператор `ALTER POLICY` используется для изменения политики безопасности.

Политики безопасности применимы только для многофакторных пользователей при многофакторном подключении к БД.

```
ALTER POLICY <имя политики> AS <параметр>=<значение> [,<параметр>=<значение>...]
```

Чтобы изменить политику безопасности пользователь должен иметь права на запись в базу данных безопасности `security5.fdb` и успешно соединиться с любой базой данных.

Возможные параметры политик можно посмотреть в описании оператора CREATE POLICY.

Подробнее о политиках безопасности см. в «Руководстве администратора».

См. также операторы CREATE POLICY, DROP POLICY.

ALTER PROCEDURE

Для изменения существующей хранимой процедуры используется оператор `ALTER PROCEDURE`. Синтаксис оператора:

Листинг Ж.15. Синтаксис оператора `ALTER PROCEDURE`

```
ALTER PROCEDURE <имя хранимой процедуры>
[AUTHID {OWNER | CALLER}]
   [(<входной пар-р>[,<входной пар-р>...])]
[RETURNS (<выходной параметр> [, <выходной параметр> ...])]
[SQL SECURITY {DEFINER | INVOKER}]
{ EXTERNAL NAME '<внешний модуль>' ENGINE <имя движка> [AS <тело внешней проц.>]} |
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
{
   {AS '<BLR код>'}
   |
   {AS
      [<объявление> [<объявление> ...] ]
   BEGIN
      <блок операторов>
   END }
}

<входной параметр> ::= <описание параметра> [{=|DEFAULT} <значение по умолчанию>]

<выходной параметр> ::= <описание параметра>

<описание параметра> ::= <имя параметра> <тип> [NOT NULL] [COLLATE <порядок
сортировки>]

<тип> ::= { <тип данных SQL>
          | [TYPE OF] <имя домена>
          | TYPE OF COLUMN <имя таблицы/представления>.<имя столбца> }

<значение по умолчанию>::= {<литерал> | NULL | <контекстная переменная>}

<внешний модуль> ::= '<имя внешнего модуля>!<имя функции в модуле>[! <информация>]'

<объявление> ::= <объявление локальной переменной>;
               | <объявление курсора>;
               | <объявление процедуры/функции>
               | <реализация процедуры/функции>
```

Изменять хранимую процедуру может ее создатель и администратор (пользователь с ролью `RDB$ADMIN`) и пользователь с привилегией `ALTER ANY PROCEDURE`.

Оператор `ALTER PROCEDURE` позволяет изменять:

- состав и характеристики входных параметров;
- состав и характеристики выходных параметров;
- в контексте какого пользователя будет выполняться процедура;
- локальные переменные, курсоры, подпрограммы;
- тело хранимой процедуры на языке SQL;
- исходный SQL код процедуры на двоичный BLR код;
- точку входа и имя движка (для внешних процедур);

Выполняемые изменения хранимой процедуры не оказывают никакого влияния на ее зависимости.

Подробнее о хранимых процедурах можно прочитать в главе 18.

См. также операторы CREATE PROCEDURE, RECREATE PROCEDURE, DROP PROCEDURE, EXECUTE PROCEDURE, DECLARE VARIABLE, DECLARE CURSOR, DECLARE FUNCTION, DECLARE PROCEDURE.

ALTER ROLE

Оператор `ALTER ROLE` изменяет список системных привилегий роли или удаляет их. Оператор `ALTER ROLE RDB$ADMIN` включает или отключает возможности администраторам Windows автоматически получать привилегии роли `RDB$ADMIN` при входе.

Листинг Ж.16. Синтаксис оператора `ALTER ROLE`

```
ALTER ROLE
{
      <имя роли> SET SYSTEM PRIVILEGES TO <сис.привилегия> [,<сис.привилегия>...]
    | <имя роли> DROP SYSTEM PRIVILEGES
    | RDB$ADMIN {SET | DROP} AUTO ADMIN MAPPING
}
```

Подробнее о работе с ролями см. в главе 26.

См. также операторы CREATE ROLE, DROP ROLE, GRANT, REVOKE.

ALTER SEQUENCE

Оператор устанавливает значение последовательности или генератора в заданное значение и/или изменяет значение приращения. Его синтаксис:

Листинг Ж.17. Синтаксис оператора `ALTER SEQUENCE`

```
ALTER SEQUENCE <имя генератора>
   [START WITH <значение>]
   [RESTART [WITH <значение>]]
   [INCREMENT [BY] <приращение>];
```

Изменить последовательность может ее владелец, администратор и пользователи с привилегией `ALTER ANY SEQUENCE (ALTER ANY GENERATOR)`.

Подробно оператор описан в главе 12.

См. также операторы CREATE GENERATOR, CREATE SEQUENCE, DROP GENERATOR, DROP SEQUENCE, SET GENERATOR, функцию GEN_ID, конструкцию NEXT VALUE FOR.

ALTER SESSION RESET

Оператор сбрасывает сеансовое окружение (подключения) к исходному состоянию. Эта функциональность полезна если сеанс используется повторно, вместо того чтобы производить отключение/-подключение.

Листинг Ж.18. Синтаксис оператора `ALTER SESSION RESET`

```
ALTER SESSION RESET
```

Данный оператор делает следующее:

- сбрасывает установленные параметры `DECFLOAT (BIND, TRAP` и `ROUND)` в значения по умолчанию;
- сбрасывает тайм-ауты сессии и оператора в 0;
- удаляет все контекстные переменные из пространства имён `USER_SESSION`;

- очищает содержимое всех используемых глобальных таблиц уровня соединения (`COMMIT PRESERVE ROWS`);
- сбрасывает роль в значение переданное в DPB (указанное при подключении) и очищает кэш привелегий (если роль была изменена с помощью оператора `SET ROLE`);
- текущая активная транзакция откатывается и стартуется новая с теми же параметрами, после рестарта.

Если в текущей активной транзакции были произведены изменения, то будет выдано предупреждение.

Если в текущей сессии активны другие транзакции, то будет выдана ошибка. При проверке транзакций перед сбросом сессии подготовленные 2PC транзакции игнорируются.

ALTER TABLE

Оператор `ALTER TABLE` используется для изменения описания таблицы, существующей в базе данных.

Синтаксис оператора `ALTER TABLE`:

Листинг Ж.19. Синтаксис оператора `ALTER TABLE`

```
ALTER TABLE <имя таблицы> <операция изменения> [, <операция изменения>...];

<операция изменения> ::= {
     ADD <определение столбца>
   | ADD <ограничение таблицы>
   | DROP <имя столбца>
   | DROP CONSTRAINT <ограничение столбца или таблицы>
   | ALTER [COLUMN] <имя столбца> <модификация столбца>
   | ALTER SQL SECURITY {DEFINER|INVOKER}
   | DROP SQL SECURITY
   | SET TABLESPACE [TO] {<имя табличного пространства> | PRIMARY}
}

<определение столбца> ::= <опр-е обычного столбца>
                        | <опр-е вычисляемого столбца>
                        | <опр-е идентификационного столбца>

<определение обычного столбца> ::=
              <имя столбца> { <тип данных> | <имя домена>}
              [DEFAULT {<литерал> | NULL | <контекстная переменная>}]
              [NOT NULL]
              [<ограничение столбца>]
              [COLLATE <порядок сортировки>]

<определение вычисляемого столбца> ::=
              <имя столбца> [<тип данных>]
             {COMPUTED [BY] | GENERATED ALWAYS AS} (<выражение>)

<определение идентификационного столбца> ::=
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
        <имя столбца> [<тип данных>]
       {ALWAYS|GENERATED BY} DEFAULT AS IDENTITY [(START WITH <начальное значение>)]
       [<ограничение столбца>]

<модификация столбца> ::= TO <новое имя столбца>
                        | POSITION <новая позиция>
                        | <мод-я обычного столбца>
                        | <мод-я вычисляемого столбца>
                        | <мод-я идентификационного столбца>

<модификация обычного столбца> ::=
             TYPE { <тип данных> | <имя домена> }
           | SET DEFAULT { <литерал> | NULL | <контекстная переменная>}
           | DROP DEFAULT
           | SET NOT NULL
           | DROP NOT NULL

<модификация вычисляемого столбца> ::=
   [TYPE <тип данных>] {GENERATED ALWAYS AS | COMPUTED [BY]} (<выражение>)

<модификация идентификационного столбца> ::=
             <опции автоинкремента>
           | SET GENERATED {ALWAYS|BY DEFAULT} [ <опции автоинкремента> ...]
           | DROP IDENTITY

<опции автоинкремента> ::= RESTART [ WITH <стартовое значение> ]
                         | SET INCREMENT [BY] <приращение>

<ограничение столбца> ::=
   [CONSTRAINT <имя ограничения>]
   { UNIQUE [<предложение USING>] [[IN] TABLESPACE {<имя табл. пространства>|PRIMARY}]
   | PRIMARY KEY [<предложение USING>] [[IN] TABLESPACE {<имя табл. прос-ва>|PRIMARY}]
   | REFERENCES <имя таблицы> [(<имя столбца>)]
         [ON DELETE { NO ACTION | CASCADE | SET DEFAULT | SET NULL }]
         [ON UPDATE { NO ACTION | CASCADE | SET DEFAULT | SET NULL }]
         [<предложение USING>] [[IN] TABLESPACE { <имя табл. пространства> | PRIMARY}]
   | CHECK (<условие столбца>)
   }

<ограничение таблицы> ::=
   [CONSTRAINT <имя ограничения>]
   { UNIQUE (<столбец> [, <столбец>...]) [<предложение USING>]
       [[IN] TABLESPACE { <имя табличного пространства> | PRIMARY}]
   | PRIMARY KEY (<столбец> [, <столбец>...]) [<предложение USING>]
       [[IN] TABLESPACE { <имя табличного пространства> | PRIMARY}]
   | FOREIGN KEY (<столбец> [, <столбец>...])
   | REFERENCES <имя таблицы> [(<столбец> [, <столбец>...])]
       [ON DELETE { NO ACTION | CASCADE | SET DEFAULT | SET NULL }]
       [ON UPDATE { NO ACTION | CASCADE | SET DEFAULT | SET NULL }]
       [<предложение USING>] [[IN] TABLESPACE { <имя табл. пространства> | PRIMARY}]
   | CHECK (<условие столбца>)
   }
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
<предложение USING> ::= USING [ASC[ENDING] | DESC[ENDING]] INDEX <имя индекса>
```

Изменять таблицу может ее владелец, администратор и пользователь с ролью `ALTER ANY TABLE`.

С помощью этого оператора можно:

- добавлять новый столбец в таблицу;
- удалять существующий столбец таблицы;
- добавлять ограничение таблицы;
- удалять существующее ограничение таблицы;
- изменять или удалять привилегии выполнения для таблицы;
- изменять имя и тип данных столбца;
- изменять позицию столбца в списке столбцов таблицы;
- добавлять или удалять значение по умолчанию столбца;
- изменять тип и выражение для вычисляемого столбца;
- изменять столбец идентификации;
- изменять табличное пространство;
- удалять табличное пространство.

Подробно оператор описан в главе 9.

См. также операторы CREATE TABLE, DROP TABLE.

ALTER TABLESPACE

Для изменения путей к файлам табличных пространств используется оператор `ALTER TABLESPACE`:

Листинг Ж.20. Синтаксис оператора изменения табличного пространства ALTER TABLESPACE

```
ALTER TABLESPACE <имя табличного пространства> SET FILE [TO] <путь к файлу>;
```

Права на изменение табличных пространств есть только у администраторов и пользователей с привилегией `ALTER ANY TABLESPACE`. Для изменения путей к файлам табличных пространств необходимо указать путь к файлу. Можно указывать как абсолютный путь, так и относительный, в том числе с использованием псевдонима директории, заданного в `directories.conf` в секции `tablespaces`. В `directories.conf` указывается реальный путь к директории с псевдонимом <псевдоним директории>. Путь к файлу табличного пространства с использованием псевдонима директории указывается в формате: `<псевдоним директории>/<файл>`, например, `dir/file.dat`. Первый компонент пути (`dir`) будет обработан как псевдоним директории. Если псевдоним не будет найден в `directories.conf`, то путь будет обработан как относительный (относительно директории, в которой размещен основной файл базы данных).

Все директории, используемые в пути, должны быть созданы заранее.

В системную таблицу `RDB$TABLESPACES` в любом случае будет записан абсолютный путь к файлу табличного пространства.

См. также операторы CREATE TABLESPACE, DROP TABLESPACE.

ALTER TRIGGER

Оператор ALTER TRIGGER позволяет изменять заголовок и/или тело существующего триггера. Синтаксис оператора:

Листинг Ж.21. Синтаксис оператора ALTER TRIGGER

```
ALTER TRIGGER <имя триггера>
[ACTIVE | INACTIVE]
[{BEFORE | AFTER} <список событий таблицы (представления)>]
[POSITION <порядок срабатывания триггера>]
[SQL SECURITY {DEFINER | INVOKER} | DROP SQL SECURITY]
[{EXTERNAL NAME '<внешний модуль>' ENGINE <имя движка> [AS <тело внеш. триг.>]} |
{
   AS
      [<объявление> [<объявление> ...] ]
   BEGIN
      <блок операторов>
   END }]

<список событий таблицы (представления)> ::= <событие DML> [OR <событие DML>...]

<событие DML> ::= { INSERT | UPDATE | DELETE }
```

DML триггер может быть изменен администратором и владельцем таблицы или представления или пользователем с привилегией ALTER ANY {TABLE | VIEW\. Триггеры для событий базы данных и триггеры событий на изменение метаданных может изменить администратор, владелец базы данных или пользователь с привилегией ALTER DATABASE.

В триггере с помощью данного оператора можно изменить:

- его состояние активности (ACTIVE / INACTIVE);
- событие (события) таблицы (представления) и фазу события;
- позицию триггера;
- выполняемые триггером действия (тело триггера);
- в контексте какого пользователя будет выполняться триггер (или удалить эту опцию);

Если какой-то элемент не указан, то его первоначальное значение не изменяется.

В этом операторе нельзя изменить ссылку на таблицу или представление, с которым связан существующий триггер, а также событие базы данных.

См. также операторы CREATE TRIGGER, RECREATE TRIGGER, CREATE OR ALTER TRIGGER, DROP TRIGGER, DECLARE VARIABLE.

ALTER USER

Для изменения существующей учетной записи пользователя используется следующий синтаксис:

Листинг Ж.22. Синтаксис оператора ALTER USER

```
ALTER {USER <логин> | CURRENT USER}
{
   [SET]
   [PASSWORD '<пароль>']
   [FIRSTNAME '<имя пользователя>']
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
    [MIDDLENAME '<отчество пользователя>']
    [LASTNAME '<фамилия пользователя>']
    [ACTIVE | INACTIVE]
    [TAGS (<атрибут>|DROP <имя атрибута> [, <атрибут>|DROP <имя атрибута>...] )]
}
[USING PLUGIN 'имя плагина']
[{GRANT | REVOKE} ADMIN ROLE];

<атрибут> ::= <имя атрибута> = 'строковое значение'
```

В операторе `ALTER USER` должно присутствовать хотя бы одно из необязательных предложений.

Это единственный оператор управления учётными записями, который может также использоваться непривилегированными пользователями для изменения их собственных учетных записей, однако это не относится к опциям `GRANT/REVOKE ADMIN ROLE` и атрибуту `ACTIVE/INACTIVE` для изменения которых, необходимы административные привилегии.

Возможность модификации учетных записей пользователей предоставлена:

- `SYSDBA`;
- Любому пользователю, имеющему права на роль `RDB$ADMIN` в базе данных пользователей и права на ту же роль для базы данных в активном подключении (пользователь должен подключаться к базе данных с ролью `RDB$ADMIN`);
- При включенном флаге `AUTO ADMIN MAPPING` в базе данных пользователей - любой администратор операционной системы Windows (при условии использования сервером доверенной авторизации) без указания роли. При этом не важно, включен или выключен флаг `AUTO ADMIN MAPPING` в самой базе данных.

Если требуется изменить свою учётную запись, то вместо указания имени текущего пользователя можно использовать предложение `CURRENT USER`.

С помощью данного оператора можно:

- поменять пароль пользователя;
- изменить дополнительные атрибуты пользователя, такие как имя пользователя (имя человека), отчество и фамилия;
- сделать учетную запись неактивной (или наоборот активировать);
- задать, изменить или удалить пользовательские атрибуты
- предоставить указанному пользователю привилегии роли `RDB$ADMIN` в базе данных безопасности.

Если предложение `USING PLUGIN` не указано, то при изменении атрибутов пользователя они сами меняется у всех соответствующих пользователей в плагинах из списка параметра `DefaultUserManagers`.

Если в каком-либо стандартном плагине нет пользователя, то он добавляется, но только есть среди изменяемых атрибутов есть пароль.

Подробнее о работе с пользователями см. в «Руководстве администратора».

См. также операторы CREATE USER, DROP USER.

ALTER VIEW

Оператор `ALTER VIEW` позволяет изменять определение представления без его пересоздания, при этом существующие права на представления и зависимости сохраняются.

Листинг Ж.23. Синтаксис оператора ALTER VIEW

```
ALTER VIEW <имя представления> [<список столбцов>]
AS <оператор SELECT>
[WITH CHECK OPTION]

<список столбцов> ::= (столбец [, столбец ...]);
```

Изменить представление могут только владелец представления, администратор, пользователь с привилегией ALTER ANY VIEW.

Подробнее об этом операторе см. в главе 15.

См. также операторы CREATE OR ALTER VIEW, CREATE VIEW, DROP VIEW, RECREATE VIEW.

CALL

Оператор CALL аналогичен EXECUTE PROCEDURE, но позволяет получать определенные выходные параметры или не получать их вовсе.

Если используется позиционная или смешанная передача, то сначала нужно указать входные параметры, а потом выходные.

Выходные параметры игнорируются, если в них передано значение NULL. В DSQL в этом случае они даже не возвращаются.

Синтаксис оператора CALL:

```
CALL [<имя пакета> .] <имя процедуры> (<параметры>)

<параметры> ::=
   <позиционные параметры> |
   [ {<позиционные параметры>,} ] <именованные параметры>

<позиционные параметры> ::=
   {<значение> | DEFAULT} [ {, <значение> | DEFAULT}... ]

<именованные параметры> ::=
   <именованные параметры> [ {, <именованные параметры>}... ]

<именованные параметры> ::=
     <значение>
   | DEFAULT
```

Примеры:

1. Создание процедуры:

```
create or alter procedure insert_customer (
   last_name varchar(30),
   first_name varchar(30)
) returns (
   id integer,
   full_name varchar(62)
)
as
begin
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
    insert into customers (last_name, first_name)
       values (:last_name, :first_name)
       returning id, last_name || ', ' || first_name
       into :id, :full_name;
end
```

2. Не все выходные параметры являются обязательными:

```
call insert_customer(
 'LECLERC',
 'CHARLES',
 ?)
```

3. Игнорирование первого выходного параметра с помощью NULL:

```
call insert_customer(
 'LECLERC',
 'CHARLES',
 null,
 ?)
```

4. Игнорировать выходной параметр ID:

```
call insert_customer(
 'LECLERC',
 'CHARLES',
 full_name => ?)
```

5. Передача входных данных и получение выходных с помощью именованных аргументов:

```
call insert_customer(
 last_name => 'LECLERC',
 first_name => 'CHARLES',
 last_name => ?,
 id => ?)
```

См. также операторы CREATE PROCEDURE, ALTER PROCEDURE, DROP PROCEDURE, SELECT.

COMMENT

Оператор COMMENT позволяет создавать примечания, комментарии для объектов базы данных.

Листинг Ж.24. Синтаксис оператора COMMENT

```
 COMMENT ON <объект> IS {'<текст>' | NULL}

<объект> ::= DATABASE
           | <базовый тип> <имя>
           | COLUMN <таблица>.<столбец>
           | [PROCEDURE | FUNCTION] PARAMETER [<пакет>.]<процедура>.<параметр>
           | {PROCEDURE | [EXTERNAL] FUNCTION} [<пакет>.]<процедура>
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
<базовый тип> ::= CHARACTER SET
                | COLLATION
                | DOMAIN
                | EXCEPTION
                | FILTER
                | GENERATOR
                | INDEX
                | PACKAGE
                | USER
                | ROLE
                | SEQUENCE
                | TABLE
                | TRIGGER
                | VIEW
```

Если текст примечания задать в виде двух подряд идущих апострофов '', то это равносильно заданию NULL, то есть удалению существующего примечания.

Добавить комментарий может администратор, владелец объекта, для которого добавляется комментарий, пользователь с привилегией ALTER ANY <тип объекта>.

COMMIT

Оператор подтверждает все изменения базы данных, выполненные в контексте текущей транзакции.

Листинг Ж.25. Синтаксис оператора COMMIT

```
COMMIT [WORK] [TRANSACTION <имя транзакции>]
[RELEASE] [RETAIN [SNAPSHOT]];
```

При выполнении оператора подтверждаются все изменения в данных, выполненные в контексте данной транзакции. Новые версии записей (измененных, добавленных, удаленных) становятся доступными для других процессов.

Подробно оператор описан в главе 5.

См. также операторы SET TRANSACTION, ROLLBACK, SAVEPOINT, RELEASE SAVEPOINT.

CONNECT

Оператор CONNECT используется для соединения с существующей базой данных. Синтаксис:

Листинг Ж.26. Синтаксис оператора CONNECT

```
  CONNECT '<спецификация файла>'
[USER '<имя пользователя>' [PASSWORD '<пароль>']]
[CACHE <целое> [BUFFERS]]
[ROLE '<имя роли>'];
```

Соединиться с базой данных может любой пользователь, описанный в системе.

Подробно оператор описан в главе 6.

См. также операторы CREATE ROLE, DROP ROLE, GRANT, REVOKE.

CREATE COLLATION

Оператор задает новый порядок сортировки для существующего в базе данных набора символов.

Листинг Ж.27. Синтаксис оператора CREATE COLLATION

```
CREATE COLLATION <имя сортировки>
FOR <имя набора символов>
[FROM <базовая сортировка> | FROM EXTERNAL ('<имя внешнего файла>')]
[NO PAD | PAD SPACE]
[CASE SENSITIVE | CASE INSENSITIVE]
[ACCENT SENSITIVE | ACCENT INSENSITIVE]
['<атрибут> [; <атрибут> ...]'];

<атрибут> ::= <имя атрибута> = <значение атрибута>
```

Оператор `CREATE COLLATION` ничего не создаёт, а делает сортировку известной для базы данных. Сортировка уже должна присутствовать в системе, как правило в файле библиотеки, и должна быть зарегистрирована в файле `fbintl.conf` подкаталога `intl` корневой директории РЕД Базы Данных.

В этом операторе можно указать:

- сортировку, на основе которой будет создана новая сортировка;
- учитываются или нет конечные пробелы при сравнении;
- будет ли сравнение чувствительно к регистру;
- будет ли сравнение чувствительно к акцентированным буквам (например «е» и «ё»);
- атрибуты для сортировки.

Создать новую сортировку может администратор и пользователь с привилегией `CREATE COLLATION`. Пользователь, создавший сортировку, становится её владельцем.

Подробно оператор описан в главе 22.

См. также оператор DROP COLLATION.

CREATE DATABASE

Оператор `CREATE DATABASE/SCHEMA` создает новую базу данных. При этом в файл базы данных помещаются многочисленные системные данные. Синтаксис оператора:

Листинг Ж.28. Синтаксис оператора CREATE DATABASE

```
CREATE {DATABASE | SCHEMA} '<спецификация файла>'
   [USER '<имя пользователя>' [PASSWORD '<пароль>'] [ROLE <имя роли>]]
   [PAGE_SIZE [=] <целое>]
   [LENGTH [=] <целое> [PAGE[S]]]
   [SET NAMES '<набор символов>']
   [DEFAULT CHARACTER SET <набор символов>
      [COLLATION <сортировка по умолчанию>]]
   [DIFFERENCE FILE '<имя файла>']
   [<вторичный файл> [<вторичный файл>...]]

<спецификация файла> ::=
            [<спецификация удалённого сервера>] { <путь к файлу БД> | <алиас БД> }

<спецификация удалённого сервера> ::=
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
                <хост>[\<порт> | <имя сервиса>]:
              | \\<хост>[@<порт> | <имя сервиса>]\
              | <протокол>://[ <имя сервиса>[: <порт> | <имя сервиса>]/]

<протокол> = inet | inet4 | inet6 | xnet

<вторичный файл> ::= FILE '<путь к файлу>'
                     [LENGTH [=] <целое> [PAGE[S]]]
                     [STARTING [AT [PAGE]] <целое>]
```

Создать новую базу данных может только администратор и пользователь с привилегией `CREATE DATABASE`.

В этом операторе можно указать:

- имя и пароль владельца базы данных;
- размер страницы базы данных;
- количество страниц в первичном файле базы данных;
- набор символов подключения;
- набор символов по умолчанию для строковых типов данных в базе данных;
- дельта-файл, который будет создаваться при выполнении команды `ALTER DATABASE BEGIN BACKUP` или запуске утилиты `nBackup` из командной строки;
- спецификацию и размер вторичных файлов;

Подробно оператор описан в главе 6.

См. также операторы ALTER DATABASE, DROP DATABASE, CREATE SHADOW, DROP SHADOW, CONNECT.

CREATE DOMAIN

Оператор `CREATE DOMAIN` создает новый домен в базе данных, с которой в настоящий момент существует соединение. Синтаксис оператора:

Листинг Ж.29. Синтаксис оператора `CREATE DOMAIN`

```
CREATE DOMAIN <имя домена> [AS] <тип данных>
   [DEFAULT {<литерал> | NULL | <контекстная переменная>}]
   [NOT NULL]
   [CHECK (<условие домена>)]
   [CHARACTER SET <набор символов> [COLLATE <порядок сортировки>] ];
```

Создавать домен может администратор и пользователь с привилегией `CREATE DOMAIN`.

В этом операторе можно указать:

- имя создаваемого домена;
- тип данных;
- значение по умолчанию;
- можно ли столбцу, основанному на этом домене, присваиваться пустое значение;
- условие, которому должно удовлетворять значение;
- набор символов и порядок сортировки при описании символьного домена.

Подробно оператор `CREATE DOMAIN` описан в главе 8.

См. также операторы ALTER DOMAIN, DROP DOMAIN.

CREATE EXCEPTION

Оператор создает пользовательское исключение. Синтаксис оператора:

Листинг Ж.30. Синтаксис оператора CREATE EXCEPTION

```
CREATE EXCEPTION <имя исключения> '<текст сообщения>';
```

Имя исключения может содержать до 63 символов и должно быть уникальным среди всех имен пользовательских исключений базы данных.

Текст сообщения — это текст, выдаваемый пользователю при вызове данного исключения. Может содержать до 1021 любых символов, включая буквы кириллицы и специальные символы. Сообщение об ошибке может содержать слоты для параметров (`@N`), которые заполняются при возбуждении исключения. Нумерация слотов начинается с 1. Максимальный номер слота равен 9.

Создать исключение может администратор и пользователь с привилегией `CREATE EXCEPTION`. Пользователь, создавший исключение, становится его владельцем.

Подробнее оператор описан в главе 16.

См. также операторы ALTER EXCEPTION, DROP EXCEPTION, RECREATE EXCEPTION, CREATE OR ALTER EXCEPTION, операторы PSQL WHEN-DO, EXCEPTION.

CREATE FUNCTION

Для создания хранимой функции используется оператор `CREATE FUNCTION`.

Листинг Ж.31. Синтаксис оператора создания хранимой функции CREATE FUNCTION

```
CREATE FUNCTION <имя хранимой функции>
   [(<входной параметр> [, <входной параметр> ...])]
RETURNS <тип> [COLLATE <сортировка>] [DETERMINISTIC]
[SQL SECURITY {DEFINER | INVOKER}]
{EXTERNAL NAME '<внешний модуль>' ENGINE <имя движка> [AS <тело внешней функции>]}|
{
   {AS '<BLR код>'}
   |
   {AS
      [<объявление> [<объявление> ...] ]
   BEGIN
      <блок операторов>
   END }
}

<входной параметр> ::= <описание параметра> [{=|DEFAULT} <значение по умолчанию>]

<описание параметра> ::= <имя параметра> <тип> [NOT NULL]
                         [COLLATE <порядок сортировки>]

<тип> ::= <тип данных SQL>
        | [TYPE OF] <имя домена>
        | TYPE OF COLUMN <имя таблицы/представления>.<имя столбца>

<значение по умолчанию> ::= {<литерал> | NULL | <контекстная переменная>}
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
<внешний модуль> ::= '<имя внешнего модуля>!<имя функции в модуле>[! <информация>]'

<объявление> ::= <объявление локальной переменной>;
                 | <объявление курсора>;
                 | <объявление процедуры/функции>
                 | <реализация процедуры/функции>
```

Хранимую функцию может создать администратор и пользователь с привилегией `CREATE FUNCTION`. Пользователь, создавший хранимую функцию, становится её владельцем.

В этом операторе можно указать:

- имя хранимой функции (должно быть уникальным среди имён всех хранимых функций и внешних (UDF) функций);
- входные параметры от вызвавшей программы;
- тип возвращаемого значения хранимой функции;
- что функция детерминированная;
- место расположения функции во внешнем модуле (внешняя функция);
- в контексте какого пользователя будет выполняться функция;
- произвольное количество локальных переменных, именованных курсоров и подпрограммы (подпроцедуры и подфункции);
- блок операторов в теле хранимой функции;
- двоичный BLR код вместо исходного SQL текста.

Подробнее оператор описан в главе 19.

См. также операторы CREATE OR ALTER FUNCTION, RECREATE FUNCTION, ALTER FUNCTION, DROP FUNCTION, DECLARE VARIABLE, DECLARE CURSOR, DECLARE FUNCTION, DECLARE PROCEDURE.

CREATE GENERATOR

Оператор `CREATE GENERATOR` используется для создания в базе данных объекта генератор (`GENERATOR` или `SEQUENCE`). Синтаксис оператора:

Листинг Ж.32. Синтаксис оператора `CREATE GENERATOR`

```
CREATE {GENERATOR | SEQUENCE} <имя генератора>
[START WITH <начальное значение>] [INCREMENT [BY] <приращение>];
```

Ключевые слова `GENERATOR` и `SEQUENCE` являются синонимами.

В этом операторе можно указать:

- имя генератора (должно быть уникальным среди имён всех генератора и должно содержать до 63 символов);
- начальное значение последовательности;
- шаг приращения для оператора `NEXT VALUES FOR` и функции `GEN_ID`.

Создавать последовательности могут администраторы и те пользователи, у кого есть привилегия `CREATE SEQUENCE (CREATE GENERATOR)`. Пользователь, создавший последовательность, становится её владельцем.

Подробно работа с генераторами описана в главе 12.

См. также операторы DROP GENERATOR, DROP SEQUENCE, CREATE SEQUENCE, SET GENERATOR, ALTER SEQUENCE, функцию GEN_ID, конструкцию NEXT VALUE FOR.

CREATE GLOBAL TEMPORARY TABLE

Временные таблицы (GTT) используются в СУБД «РЕД База Данных» для хранения данных, которые относятся к одной сессии или одной транзакции. Временные таблицы (GLOBAL TEMPORARY) в отличие от постоянных таблиц целесообразно использовать в тех случаях, когда сохраняемые данные часто изменяются, и непостоянны, и предназначены только для хранения временных для некоторой сессии данных.

Листинг Ж.33. Синтаксис оператора создания глобальной временной таблицы

```
CREATE GLOBAL TEMPORARY TABLE <имя таблицы>
   (<определение столбца> [, {<определение столбца> | <ограничение таблицы>}...]);
   [ON COMMIT {DELETE | PRESERVE} ROWS]
   [SQL SECURITY {DEFINER | INVOKER}];
```

Если в операторе создания глобальной временной таблицы указано необязательное предложение `ON COMMIT DELETE ROWS`, то будет создана GTT транзакционного уровня (по умолчанию). При указании предложения `ON COMMIT PRESERVE ROWS` будет создана GTT уровня соединения с базой данных.

Более подробное описание GTT можно найти в главе 9.

См. также операторы CREATE TABLE, ALTER TABLE, DROP TABLE.

CREATE INDEX

Оператор позволяет создать индекс для таблицы базы данных. Его синтаксис:

Листинг Ж.34. Синтаксис оператора `CREATE INDEX`

```
CREATE [UNIQUE] [ASC[ENDING] | DESC[ENDING]]
INDEX <имя индекса> ON <таблица>
{(<столбец> [, <столбец> ...]) | COMPUTED BY (<выражение>)}
[WHERE <условие>]
[[IN] TABLESPACE {<имя табличного пространства> | PRIMARY}];
```

Индекс создается для одной конкретной таблицы.

Создать индекс может только владелец таблицы, для которой создан индекс, администратор и пользователь с привилегией `ALTER ANY TABLE`.

В этом операторе можно указать:

- имя индекса (должно быть уникальным среди имен всех индексов базы данных);
- уникален ли индекс;
- записи индекса упорядочиваются по возрастанию или убыванию значений столбцов;
- выражение в вычисляемом индексе;
- условие для создания частичного индекса только по подходящим записям таблицы;
- табличное пространство индекса.

Подробно оператор описан в главе 14.

См. также операторы ALTER INDEX, DROP INDEX, SET STATISTICS.

CREATE JOB

Оператор CREATE JOB позволяет создавать задания для планировщика.

Листинг Ж.35. Синтаксис оператора CREATE JOB

```
CREATE JOB <имя_задания>
<параметры расписания>
[ACTIVE | INACTIVE]
[START DATE '<timestamp>' | NULL]
[END DATE '<timestamp>' | NULL]
{AS
    <объявления переменных>
BEGIN
    <блок sql-операторов>
END
| COMMAND '<команда>' }

<параметры расписания> ::= '<Минуты> <Часы> <Дни_месяца> <Месяцы> <Дни_недели>'
```

Задание выполняется от имени пользователя, который его создал.

Создать задание может пользователь с административными привилегиями или с привилегией CREATE JOB.

При использовании данного оператора можно указать:

- имя задания;
- его активность (будет ли задание запускаться или нет);
- расписание запуска;
- интервал времени, в котором запускается задание;
- команды ОС;
- локальные переменные, курсоры, подпрограммы;
- тело задания;

Подробно о планировщике заданий описано в главе 23.

См. также операторы DROP JOB, ALTER JOB.

CREATE MAPPING

Оператор CREATE MAPPING создаёт отображение объектов безопасности (пользователей, групп, ролей) одного или нескольких плагинов аутентификации на внутренние объекты безопасности – CURRENT_USER и CURRENT_ROLE.

Листинг Ж.36. Синтаксис оператора CREATE MAPPING

```
CREATE [GLOBAL] MAPPING <имя отображения>
USING {
   PLUGIN <имя плагина> [IN <имя базы данных>]
 | ANY PLUGIN [IN <имя базы данных> | SERVERWIDE]
 | MAPPING [IN <имя базы данных>]
 | '*' [IN <имя базы данных>] }
FROM { ANY <тип отображаемого объекта> | <тип отображаемого объекта> <имя
отображаемого объекта> }
TO { USER | ROLE } [<имя объекта, на которое произведено отображение>]
```

Воспользоваться оператором создания отображений может SYSDBA, владелец базы данных (если отображение локальное), пользователь с ролью RDB$ADMIN, пользователь root (Linux).

При использовании данного оператора можно указать:

- что отображение будет применено не только для текущей базы данных, но и для всех баз данных находящимся в том же кластере (GLOBAL);
- имя отображения;
- источник отображения;
- отображаемый объект;
- пользователя или роль, на которого будет произведено отображение.

Подробнее синтаксис оператора разобран в главе 26.

См. также операторы CREATE OR ALTER MAPPING, DROP MAPPING, ALTER MAPPING.

CREATE OR ALTER EXCEPTION

Оператор создает новое пользовательское исключение, если оно отсутствует в базе данных, или изменяет существующее, при этом существующие зависимости исключения будут сохранены. Синтаксис оператора:

Листинг Ж.37. Синтаксис оператора CREATE OR ALTER EXCEPTION

```
CREATE OR ALTER EXCEPTION <имя исключения> '<текст сообщения>';
```

Имя исключения может содержать до 63 символов и должно быть уникальным среди всех имен исключений базы данных.

Текст сообщения — текст, выдаваемый в момент вызова исключения. Может содержать до 1021 символа.

Подробнее об исключениях см. в главе 16.

См. также операторы ALTER EXCEPTION, DROP EXCEPTION, CREATE EXCEPTION, RECREATE EXCEPTION, операторы PSQL WHEN-DO, EXCEPTION.

CREATE OR ALTER FUNCTION

Оператор CREATE OR ALTER FUNCTION позволяет создать новую хранимую функцию, если функция с тем же именем отсутствует в базе данных, или изменить описание существующей в базе данных функции. Если функция с этим именем уже существует, то происходит ее замена на новую хранимую функцию, при этом существующие привилегии и зависимости сохраняются. Синтаксис оператора:

Листинг Ж.38. Синтаксис оператора создания новой или изменения существующей хранимой функции CREATE OR ALTER FUNCTION

```
CREATE OR ALTER FUNCTION <имя хранимой функции>
   [(<входной параметр> [, <входной параметр> ...])]
RETURNS <тип> [COLLATE <сортировка>] [DETERMINISTIC]
[SQL SECURITY {DEFINER | INVOKER}]
{ EXTERNAL NAME '<внешний модуль>' ENGINE <имя движка> [AS <тело внешней функции>]}
|
{
   {AS '<BLR код>'}
   |{AS
      [<объявление> [<объявление> ...] ]
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
   BEGIN
      <блок операторов>
   END }
}
```

Семантика операторов и предложений в этом операторе полностью соответствует оператору `CREATE FUNCTION`.

См. также операторы CREATE FUNCTION, RECREATE FUNCTION, ALTER FUNCTION, DROP FUNCTION, DECLARE VARIABLE, DECLARE CURSOR, DECLARE FUNCTION, DECLARE PROCEDURE.

CREATE OR ALTER MAPPING

Оператор `CREATE OR ALTER MAPPING` создаёт новое или изменяет существующее отображение. Если отображение не существует, то оно будет создано с использованием предложения `CREATE MAPPING`.

Листинг Ж.39. Синтаксис оператора `CREATE OR ALTER MAPPING`

```
CREATE OR ALTER [GLOBAL] MAPPING <имя отображения>
USING {
   PLUGIN <имя плагина> [IN <имя базы данных>]
 | ANY PLUGIN [IN <имя базы данных> | SERVERWIDE]
 | MAPPING [IN <имя базы данных>]
 | '*' [IN <имя базы данных>] }
FROM { ANY <тип отображаемого объекта> | <тип отображаемого объекта> <имя
отображаемого объекта> }
TO { USER | ROLE } [<имя объекта, на которое произведено отображение>]
```

Семантика операторов и предложений в этом операторе полностью соответствует оператору `CREATE MAPPING`.

См. также операторы CREATE MAPPING, DROP MAPPING, ALTER MAPPING.

CREATE OR ALTER PACKAGE

Оператор `CREATE OR ALTER PACKAGE` создаёт новый или изменяет существующий заголовок пакета. Синтаксис оператора:

Листинг Ж.40. Синтаксис оператора `CREATE OR ALTER PACKAGE`

```
CREATE OR ALTER PACKAGE <имя пакета>
[SQL SECURITY {DEFINER | INVOKER}]
AS
BEGIN
   [ <объявление процедуры> | <объявление функции> ...]
END

<объявление процедуры> ::=
   PROCEDURE <имя процедуры> [(<входной параметр> [, <входной параметр> ...])]
      [RETURNS (<выходной параметр> [, <выходной параметр> ...])]

<объявление функции> ::=
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
   FUNCTION <имя функции> [(<входной параметр> [, <входной параметр> ...])]
      RETURNS <тип> [COLLATE <сортировка>] [DETERMINISTIC]

<входной параметр> ::= <описание параметра> [{=|DEFAULT} <значение по умолчанию>]

<выходной параметр> ::= <описание параметра>

<описание параметра> ::= <имя параметра> <тип> [NOT NULL]
                         [COLLATE <порядок сортировки>]

<тип> ::= <тип данных SQL>
        | [TYPE OF] <имя домена>
        | TYPE OF COLUMN <имя таблицы/представления>.<имя столбца>

<значение по умолчанию>::= {<литерал> | NULL | <контекстная переменная>}
```

Если заголовок пакета не существует, то он будет создан с использованием предложения `CREATE PACKAGE`. Если он уже существует, то он будет изменен и перекомпилирован, при этом существующие привилегии и зависимости сохраняются.

См. также операторы CREATE PACKAGE BODY, RECREATE PACKAGE, DROP PACKAGE, ALTER PACKAGE, CREATE PACKAGE, CREATE PROCEDURE, CREATE FUNCTION.

CREATE OR ALTER PROCEDURE

Оператор `CREATE OR ALTER PROCEDURE` позволяет создать новую хранимую процедуру, если процедура с тем же именем отсутствует в базе данных. Если такая процедура уже существует, то происходит ее замена на новую. Синтаксис оператора:

Листинг Ж.41. Синтаксис оператора `CREATE OR ALTER PROCEDURE`

```
CREATE OR ALTER PROCEDURE <имя хранимой процедуры>
[AUTHID {OWNER | CALLER}]
   [(<входной параметр> [, <входной параметр> ...])]
[RETURNS (<выходной параметр> [, <выходной параметр> ...])]
[SQL SECURITY {DEFINER | INVOKER}]
{ EXTERNAL NAME '<внешний модуль>' ENGINE <имя движка> [AS <тело внешней проц.>]} |
{
   {AS '<BLR код>'}
   |
   {AS
      [<объявление> [<объявление> ...] ]
   BEGIN
      <блок операторов>
   END }
}

<входной параметр> ::= <описание параметра> [{=|DEFAULT} <значение по умолчанию>]

<выходной параметр> ::= <описание параметра>

<описание параметра> ::= <имя параметра> <тип> [NOT NULL] [COLLATE <порядок
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
сортировки>]

<тип> ::= { <тип данных SQL>
          | [TYPE OF] <имя домена>
          | TYPE OF COLUMN <имя таблицы/представления>.<имя столбца> }

<значение по умолчанию>::= {<литерал> | NULL | <контекстная переменная>}

<внешний модуль> ::= '<имя внешнего модуля>!<имя функции в модуле>[! <информация>]'

<объявление> ::= <объявление локальной переменной>;
                | <объявление курсора>;
                | <объявление процедуры/функции>
                | <реализация процедуры/функции>
```

Семантика операторов и предложений в этом операторе полностью соответствует оператору `CREATE PROCEDURE`.

См. также операторы CREATE PROCEDURE, RECREATE PROCEDURE, ALTER PROCEDURE, DROP PROCEDURE, EXECUTE PROCEDURE, DECLARE VARIABLE, DECLARE CURSOR, DECLARE FUNCTION, DECLARE PROCEDURE.

CREATE OR ALTER SEQUENCE

С помощью оператора `CREATE OR ALTER GENERATOR (SEQUENCE)` можно создать новую или изменить существующую последовательность:

Листинг Ж.42. Синтаксис оператора `CREATE OR ALTER GENERATOR /SEQUENCE`

```
CREATE OR ALTER {GENERATOR | SEQUENCE} <имя генератора>
   [{START WITH <начальное значение> | RESTART}]
   [INCREMENT [BY] <приращение>];
```

Если последовательности не существует, то она будет создана. Уже существующая последовательность будет изменена, при этом существующие зависимости последовательности будут сохранены.

Семантика операторов и предложений в этом операторе полностью соответствует оператору `CREATE GENERATOR`.

См. также операторы CREATE SEQUENCE, DROP GENERATOR, SET GENERATOR, ALTER SEQUENCE, функцию GEN_ID, конструкцию NEXT VALUE FOR.

CREATE OR ALTER TRIGGER

Оператор `CREATE OR ALTER TRIGGER` создает новый триггер, если триггера с таким именем не существует в базе данных. Иначе существующий триггер заменяется на новый синтаксис оператора:

Листинг Ж.43. Синтаксис оператора `CREATE OR ALTER TRIGGER`

```
CREATE OR ALTER TRIGGER <имя триггера> {
   <объявление табличного триггера>
 | <объявление табличного триггера в стандарте SQL-2003>
 | <объявление триггера базы данных>
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
 | <объявление DDL триггера> }
[SQL SECURITY {DEFINER | INVOKER}]
{ EXTERNAL NAME '<внешний модуль>' ENGINE <имя движка> [AS <тело внеш. триг.>] } |
{
   AS
      [<объявление> [<объявление> ...] ]
   BEGIN
      <блок операторов>
   END }

<объявление табличного триггера> ::=
         FOR {<имя таблицы> | <имя представления>}
         [ACTIVE | INACTIVE]
         {BEFORE | AFTER} <список событий таблицы (представления)>
         [POSITION <порядок срабатывания триггера>]

<объявление табличного триггера в стандарте SQL-2003> ::=
         [ACTIVE | INACTIVE]
         {BEFORE | AFTER} <список событий таблицы (представления)>
         [POSITION <порядок срабатывания триггера>]
         ON {<имя таблицы> | <имя представления>}

<объявление триггера базы данных> ::=
         [ACTIVE | INACTIVE]
         ON <событие соединения или транзакции>
         [POSITION <порядок срабатывания триггера>]

<объявление DDL триггера> ::=
         [ACTIVE | INACTIVE]
         {BEFORE | AFTER} <список DDL событий>
         [POSITION <порядок срабатывания триггера>]

<список событий таблицы (представления)> ::= <событие DML> [OR <событие DML>...]

<событие DML> ::= { INSERT | UPDATE | DELETE }

<событие соединения или транзакции> ::= {
           CONNECT
         | DISCONNECT
         | TRANSACTION START
         | TRANSACTION COMMIT
         | TRANSACTION ROLLBACK }

<список DDL событий> ::= {
           ANY DDL STATEMENT
         | <DDL событие> [OR <DDL событие> ...] }

<DDL событие> ::=
           CREATE|ALTER|DROP TABLE
         | CREATE|ALTER|DROP PROCEDURE
         | CREATE|ALTER|DROP FUNCTION
         | CREATE|ALTER|DROP TRIGGER
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
          | CREATE|ALTER|DROP EXCEPTION
          | CREATE|ALTER|DROP VIEW
          | CREATE|ALTER|DROP DOMAIN
          | CREATE|ALTER|DROP ROLE
          | CREATE|ALTER|DROP SEQUENCE
          | CREATE|ALTER|DROP USER
          | CREATE|ALTER|DROP INDEX
          | CREATE|DROP COLLATION
          | ALTER CHARACTER SET
          | CREATE|ALTER|DROP PACKAGE
          | CREATE|DROP PACKAGE BODY
          | CREATE|ALTER|DROP MAPPING

<объявление> ::= <объявление локальной переменной>;
               | <объявление курсора>;
               | <объявление процедуры/функции>
               | <реализация процедуры/функции>
```

Семантика операторов и предложений в этом операторе полностью соответствует оператору `CREATE TRIGGER`.

См. также операторы CREATE TRIGGER, RECREATE TRIGGER, ALTER TRIGGER, DROP TRIGGER, DECLARE VARIABLE.

CREATE OR ALTER USER

Можно управлять учётными записями пользователей средствами операторов SQL. Для создания или изменения существующей учетной записи пользователя используется следующий синтаксис:

Листинг Ж.44. Синтаксис оператора `CREATE OR ALTER USER`

```
CREATE OR ALTER USER <логин>
{
   [SET]
   [PASSWORD '<пароль>']
   [FIRSTNAME '<имя пользователя>']
   [MIDDLENAME '<отчество пользователя>']
   [LASTNAME '<фамилия пользователя>']
   [ACTIVE | INACTIVE]
   [TAGS (<атрибут>|DROP <имя атрибута> [, <атрибут>|DROP <имя атрибута>...] )]
}
[USING PLUGIN 'имя плагина']
[{GRANT | REVOKE} ADMIN ROLE];

<атрибут> ::= <имя атрибута> = 'строковое значение'
```

Если предложение `USING PLUGIN` не указано, то при создании пользователя он сам добавляется во все плагины из списка параметра `DefaultUserManagers` (в том числе его атрибуты).

При изменении пароля пользователя он сам меняется у всех плагинов из списка параметра `UserManager`. Если в каком-то плагине нет пользователя, то он добавляется.

Если меняется какой-либо другой атрибут, то он также меняется и в других плагинах, но если пользователь отсутствует, то он не создаётся.

См. также операторы CREATE USER, ALTER USER.

CREATE OR ALTER VIEW

Оператор `CREATE OR ALTER VIEW` изменяет определение представления (как `ALTER VIEW`), если оно существует, или создает его, если оно не существует.

Листинг Ж.45. Синтаксис оператора `CREATE OR ALTER VIEW`

```
CREATE OR ALTER VIEW <имя представления> [<список столбцов>]
AS <оператор SELECT>
[WITH CHECK OPTION]

<список столбцов> ::= (столбец [, столбец ...])
```

Все предложения в этом операторе в точности соответствуют предложениям в операторе `CREATE VIEW`.

См. также операторы CREATE VIEW, RECREATE VIEW, DROP VIEW, ALTER VIEW.

CREATE PACKAGE

Оператор `CREATE PACKAGE` создаёт новый заголовок пакета. Синтаксис оператора:

Листинг Ж.46. Синтаксис оператора создания заголовка пакета `CREATE PACKAGE`

```
CREATE PACKAGE <имя пакета>
[SQL SECURITY {DEFINER | INVOKER}]
AS
BEGIN
   [ <объявление процедуры> | <объявление функции> ...]
END

<объявление процедуры> ::=
   PROCEDURE <имя процедуры> [(<входной параметр> [, <входной параметр> ...])]
      [RETURNS (<выходной параметр> [, <выходной параметр> ...])]

<объявление функции> ::=
   FUNCTION <имя функции> [(<входной параметр> [, <входной параметр> ...])]
      RETURNS <тип> [COLLATE <сортировка>] [DETERMINISTIC]

<входной параметр> ::= <описание параметра> [{=|DEFAULT} <значение по умолчанию>]

<выходной параметр> ::= <описание параметра>

<описание параметра> ::= <имя параметра> <тип> [NOT NULL]
                         [COLLATE <порядок сортировки>]

<тип> ::= <тип данных SQL>
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
        | [TYPE OF] <имя домена>
        | TYPE OF COLUMN <имя таблицы/представления>.<имя столбца>

<значение по умолчанию>::= {<литерал> | NULL | <контекстная переменная>}
```

Создать новый заголовок пакета может только администратор и пользователь с привилегией `CREATE PACKAGE`. Пользователь, создавший заголовок пакета становится владельцем пакета.

В этом операторе можно:

- указать имя пакета (должно быть уникальным среди имён всех пакетов);
- задать в контексте какого пользователя будет выполняться пакет;
- объявить процедуры и функции (доступны вне тела пакета через полный идентификатор имён процедур и функций (`<имя пакета>.<имя процедуры>` и `<имя пакета>.<имя функции>`)).

Подробнее о пакетах описано в главе 21.

См. также операторы CREATE PACKAGE BODY, RECREATE PACKAGE, DROP PACKAGE, ALTER PACKAGE, CREATE OR ALTER PACKAGE, CREATE PROCEDURE, CREATE FUNCTION.

CREATE PACKAGE BODY

Оператор `CREATE PACKAGE BODY` создаёт новое тело пакета. Тело пакета может быть создано только после того как будет создан заголовок пакета. Если заголовка пакета с именем `<имя пакета>` не существует, то будет выдана соответствующая ошибка. Синтаксис оператора:

Листинг Ж.47. Синтаксис оператора создания тела пакета `CREATE PACKAGE BODY`

```
CREATE PACKAGE BODY <имя пакета>
AS
BEGIN
   [ <объявление процедуры> | <объявление функции> ...]
   [ <реализация процедуры> | <реализация функции> ...]
END

<объявление процедуры> ::=
   PROCEDURE <имя процедуры> [(<входной параметр> [, <входной параметр> ...])]
      [RETURNS (<выходной параметр> [, <выходной параметр> ...])]

<объявление функции> ::=
   FUNCTION <имя функции> [(<входной параметр> [, <входной параметр> ...])]
      RETURNS <тип> [COLLATE <сортировка>] [DETERMINISTIC]

<реализация процедуры> ::=
   PROCEDURE <имя процедуры> [(<входной_параметр> [, <входной_параметр> ...])]
      [RETURNS (<выходной параметр> [, <выходной параметр> ...])]
      {EXTERNAL NAME '<внешний модуль>' ENGINE <имя движка> [AS <тело вн. проц.>]} |
      { AS
         [<объявление> [<объявление> ...] ]
      BEGIN
         <блок операторов>
      END }

<реализация функции> ::=
   FUNCTION <имя функции> [(<входной_параметр> [, <входной_параметр> ...])]
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
      RETURNS <тип> [COLLATE <сортировка>] [DETERMINISTIC]
      {EXTERNAL NAME '<внешний модуль>' ENGINE <имя движка> [AS <тело вн. функ.>]} |
      { AS
         [<объявление> [<объявление> ...] ]
      BEGIN
         <блок операторов>
      END }

<входной параметр> ::= <описание параметра> [{=|DEFAULT} <значение по умолчанию>]

<выходной параметр> ::= <описание параметра>

<описание параметра> ::= <имя параметра> <тип> [NOT NULL]
                         [COLLATE <порядок сортировки>]

<тип> ::= {
   <тип данных SQL>
 | [TYPE OF] <имя домена>
 | TYPE OF COLUMN <имя таблицы/представления>.<имя столбца> }

<значение по умолчанию>::= {<литерал> | NULL | <контекстная переменная>}

<внешний модуль> ::= '<имя внешнего модуля>!<имя функции в модуле>[!<информация>]'

<объявление> ::= <объявление локальной переменной>;
               | <объявление курсора>;
               | <объявление процедуры/функции>
               | <реализация процедуры/функции>
```

Выполнить оператор создания тела пакета может администратор, владелец пакета или пользователь с привилегией `CREATE PACKAGE`.

Все процедуры и функции, объявленные в заголовке пакета, должны быть реализованы в теле пакета с той же сигнатурой. Кроме того, должны быть реализованы и все процедуры и функции, объявленные в теле пакета, с той же сигнатурой. Процедуры и функции, определенные в теле пакета, но не объявленные в заголовке пакета, не видны вне тела пакета.

Имена процедур и функций, объявленные в теле пакета, должны быть уникальны среди имён процедур и функций, объявленных в заголовке и теле пакета.

Значения по умолчанию для параметров процедур не могут быть переопределены. Это означает, что они могут быть в реализации только для частных процедур, которые не были объявлены.

Подробнее о пакетах описано в главе 21.

См. также операторы CREATE PACKAGE, RECREATE PACKAGE BODY, DROP PACKAGE BODY, CREATE PROCEDURE, CREATE FUNCTION.

CREATE POLICY

Для создание политики используется оператор `CREATE POLICY`. Синтаксис этого оператора приведен ниже:

```
CREATE POLICY <имя политики> [AS <параметр>=<значение> [,<параметр>=<значение>...]]
```

Политики безопасности работают со всеми известными методами аутентификации.

Хотя сами политики хранятся в базе данных безопасности `security5.fdb`, создать их можно, соединившись с любой базой данных. Однако пользователь при этом должен иметь права на запись в базу данных безопасности `security5.fdb`.

Возможные параметры политик следующие:

- `AUTH_FACTORS` — факторы аутентификации:
  - `SRP` — пароль (верификатор) SRP
  - `LEGACY` — пароль Legacy
  - `WIN_SSPI`
  - `GSS`
  - `CERTIFICATE` — сертификат пользователя
  - `GOSTPASSWORD` — пароль по ГОСТ
- `PSWD_NEED_CHAR` — минимальное количество букв в пароле;
- `PSWD_NEED_DIGIT` — минимальное количество цифр в пароле;
- `PSWD_NEED_DIFF_CASE` — требование использования различных регистров букв в пароле;
- `PSWD_MIN_LEN` — минимальная длина пароля;
- `PSWD_VALID_DAYS` — срок действия пароля;
- `PSWD_UNIQUE_COUNT` — количество последних не повторяющихся паролей;
- `MAX_FAILED_COUNT` — количество неудачных попыток входа;
- `MAX_SESSIONS` — максимальное число одновременных подключений одного пользователя;
- `MAX_IDLE_TIME` — время бездействия, в секундах;
- `MAX_UNUSED_DAYS` — максимальное время неактивности учетных записей пользователя, в днях.

Пример создания политики см. в главе 26.

Для удобного просмотра всех созданных политик существует псевдотаблица безопасности `SEC$POLICIES` (см. Руководство администратора Приложение Б "Псевдотаблицы безопасности").

См. также операторы ALTER POLICY, DROP POLICY.

CREATE PROCEDURE

Для создания хранимой процедуры используется оператор `CREATE PROCEDURE`. Синтаксис оператора:

Листинг Ж.48. Синтаксис оператора `CREATE PROCEDURE`

```
CREATE PROCEDURE <имя хранимой процедуры>
[AUTHID {OWNER | CALLER}]
   [(<входной параметр> [, <входной параметр> ...])]
[RETURNS (<выходной параметр> [, <выходной параметр> ...])]
[SQL SECURITY {DEFINER | INVOKER}]
{ EXTERNAL NAME '<внешний модуль>' ENGINE <имя движка> [AS <тело внешней проц.>] }
|
{
    {AS '<BLR код>'}
    |
    {AS
      [<объявление> [<объявление> ...] ]
    BEGIN
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
      <блок операторов>
    END }
}

<входной параметр> ::= <описание параметра> [{=|DEFAULT} <значение по умолчанию>]

<выходной параметр> ::= <описание параметра>

<описание параметра> ::= <имя параметра> <тип> [NOT NULL] [COLLATE <порядок
сортировки>]

<тип> ::= { <тип данных SQL>
          | [TYPE OF] <имя домена>
          | TYPE OF COLUMN <имя таблицы/представления>.<имя столбца> }

<значение по умолчанию>::= {<литерал> | NULL | <контекстная переменная>}

<внешний модуль> ::= '<имя внешнего модуля>!<имя функции в модуле>[! <информация>]'

<объявление> ::= <объявление локальной переменной>;
               | <объявление курсора>;
               | <объявление процедуры/функции>
               | <реализация процедуры/функции>
```

Хранимую процедуру может создать администратор и пользователь с привилегией `CREATE PROCEDURE`.

В операторе `CREATE PROCEDURE` можно задать:

- имя хранимой процедуры (может содержать до 63 символов и должно быть уникальным среди имен хранимых процедур базы данных, таблиц и представлений);
- входные параметры;
- выходные параметры;
- в контексте какого пользователя будет выполняться процедура;
- локальные переменные, курсоры, подпрограммы;
- тело хранимой процедуры;
- двоичный BLR код вместо исходного SQL текста;
- точку входа и имя движка (для внешних процедур).

Подробнее о процедурах описано в главе 18.

См. также операторы CREATE OR ALTER PROCEDURE, RECREATE PROCEDURE, ALTER PROCEDURE, DROP PROCEDURE, EXECUTE PROCEDURE, SELECT, DECLARE VARIABLE, DECLARE CURSOR, DECLARE FUNCTION, DECLARE PROCEDURE.

CREATE ROLE

Оператор позволяет создать в базе данных безопасности новую роль. Синтаксис оператора:

Листинг Ж.49. Синтаксис оператора `CREATE ROLE`

```
CREATE ROLE <имя роли>
[SET SYSTEM PRIVILEGES TO <сис.привилегия> [,<сис.привилегия> ...]];
```

Имя роли может содержать до 63 символов и должно быть уникальным среди всех имен ролей.

Создать новую роль может администратор и пользователь с привилегией `CREATE ROLE`.

Предложение `SET SYSTEM PRIVILEGES TO` позволяет создать роль с системными привилегиями.

Подробнее об операторах управления ролями см. в главе 26.

См. также операторы DROP ROLE, GRANT, REVOKE, CONNECT.

CREATE SEQUENCE

Другое название для оператора создания генератора. См. в этом приложении CREATE GENERATOR.

Листинг Ж.50. Синтаксис оператора `CREATE SEQUENCE`

```
CREATE {GENERATOR | SEQUENCE} <имя генератора>;
```

Создавать последовательности могут администраторы и те пользователи, у кого есть привилегия `CREATE SEQUENCE (CREATE GENERATOR)`. Пользователь, создавший последовательность, становится её владельцем.

Подробно работа с генераторами описана в главе 12.

См. также операторы CREATE GENERATOR, DROP SEQUENCE, SET GENERATOR, ALTER SEQUENCE, функцию GEN_ID, конструкцию NEXT VALUE FOR.

CREATE SHADOW

Оператор `CREATE SHADOW` используется для создания новой оперативной копии для базы данных. Его синтаксис:

Листинг Ж.51. Синтаксис оператора `CREATE SHADOW`

```
CREATE SHADOW <номер оперативной копии> [AUTO | MANUAL][CONDITIONAL]
              '<спецификация файла>' [LENGTH [=] <целое> [PAGE[S]]]
              [<вторичный файл>]...;

<вторичный файл> ::= FILE '<спецификация файла>'
                     [LENGTH [=] <целое> [PAGE[S]]]
                     [STARTING [AT [PAGE]] <целое>]
```

Создать оперативную копию может владелец базы данных, администратор и пользователь с привилегией `ALTER DATABASE`.

В этом операторе можно указать:

- номер оперативной копии — положительное число, идентифицирующее набор файлов данной оперативной копии;
- режим работы базы данных, если оперативная копия становится недоступной;
- условную оперативную копию, которая заменяет бывшую активной перед этим оперативную копию, которая стала выполнять роль основной базы данных;
- размер оперативной копии в страницах базы данных;
- вторичные файлы.

Подробно оператор описан в главе 7.

См. также операторы CREATE DATABASE, ALTER DATABASE, DROP DATABASE, DROP SHADOW.

CREATE TABLE

Оператор CREATE TABLE создает новую таблицу в базе данных. Синтаксис оператора:

Листинг Ж.52. Синтаксис оператора CREATE TABLE

```
CREATE [GLOBAL TEMPORARY] TABLE <имя таблицы>
   [EXTERNAL [FILE] '<спецификация файла>' [ADAPTER 'CSV']]
   (<определение столбца> [, { <определение столбца> | <ограничение таблицы>}...])
   [ON COMMIT {DELETE | PRESERVE} ROWS]
   [SQL SECURITY {DEFINER | INVOKER}];
   [[IN] TABLESPACE {<имя табличного пространства> | PRIMARY}];

<определение столбца> ::= { <опр-е обычного столбца>
                          | <опр-е вычисляемого столбца>
                          | <опр-е идентификационного столбца> }

<определение обычного столбца> ::=
             <имя столбца> { <тип данных> | <имя домена>}
             [DEFAULT {<литерал> | NULL | <контекстная переменная>}]
             [NOT NULL]
             [<ограничение столбца>]
             [COLLATE <порядок сортировки>]

<определение вычисляемого столбца> ::=
             <имя столбца> [<тип данных>]
             {COMPUTED [BY] | GENERATED ALWAYS AS} (<выражение>)

<определение идентификационного столбца> ::=
       <имя столбца> [<тип данных>]
       GENERATED {ALWAYS|BY DEFAULT} AS IDENTITY [(<опции ав.> [<опции ав.>])]
       [<ограничение столбца>]

<опции автоинкремента> ::= START WITH <начальное значение>
                         | INCREMENT [BY] <приращение>

<ограничение столбца> ::=
  [CONSTRAINT <имя ограничения>]
  { UNIQUE [<предложение USING>] [[IN] TABLESPACE {<имя табл.пространства>|PRIMARY}]
  | PRIMARY KEY [<предложение USING>] [[IN] TABLESPACE {<имя табл.прос-ва>|PRIMARY}]
   | REFERENCES <имя таблицы> [(<имя столбца>)]
      [ON DELETE { NO ACTION | CASCADE | SET DEFAULT | SET NULL }]
      [ON UPDATE { NO ACTION | CASCADE | SET DEFAULT | SET NULL }]
      [<предложение USING>] [[IN] TABLESPACE { <имя табл. пространства> | PRIMARY}]
   | CHECK (<условие столбца>)
   }

<ограничение таблицы> ::=
   [CONSTRAINT <имя ограничения>]
   { UNIQUE (<столбец> [, <столбец>...]) [<предложение USING>]
        [[IN] TABLESPACE { <имя табличного пространства> | PRIMARY}]
   | PRIMARY KEY (<столбец> [, <столбец>...]) [<предложение USING>]
        [[IN] TABLESPACE { <имя табличного пространства> | PRIMARY}]
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
    | FOREIGN KEY (<столбец> [, <столбец>...])
    | REFERENCES <имя таблицы> [(<столбец> [, <столбец>...])]
       [ON DELETE { NO ACTION | CASCADE | SET DEFAULT | SET NULL }]
       [ON UPDATE { NO ACTION | CASCADE | SET DEFAULT | SET NULL }]
       [<предложение USING>] [[IN] TABLESPACE { <имя табл. пространства> | PRIMARY}]
    | CHECK (<условие столбца>)
}

<предложение USING> ::= USING [ASC[ENDING] | DESC[ENDING]] INDEX <имя индекса>
```

Создать новую таблицу может администратор и пользователь с привилегией `CREATE TABLE`. Пользователь, создавший таблицу, становится её владельцем.

С помощью данного оператора можно:

- указать имя таблицы (должно быть уникальным среди имен таблиц базы данных, хранимых процедур и представлений, описанных в этой базе данных);
- создать глобальную временную таблицу;
- использовать внешние файлы;
- добавлять столбцы в таблицу, в том числе вычисляемые и идентификационные;
- добавлять ограничение таблицы;
- изменять привилегии выполнения для таблицы;

Подробно оператор описан в главе 9.

См. также операторы ALTER TABLE, DROP TABLE.

CREATE TABLESPACE

Для создания табличного пространства используется оператор `CREATE TABLESPACE`:

Листинг Ж.53. Синтаксис оператора создания табличного пространства `CREATE TABLESPACE`

```
CREATE TABLESPACE <имя табличного пространства> FILE <путь к файлу>;
```

Права на создание табличных пространств есть только у администраторов и пользователей с привилегией `CREATE TABLESPACE`. Максимально доступно 254 табличных пространства.

Для создания табличного пространства необходимо указать путь к файлу. Можно указывать как абсолютный путь, так и относительный, в том числе с использованием псевдонима директории, заданного в `directories.conf` в секции `tablespaces`. В `directories.conf` указывается реальный путь к директории с псевдонимом `<псевдоним директории>`. Путь к файлу табличного пространства с использованием псевдонима директории указывается в формате: `<псевдоним директории>/<файл>`, например, `dir/file.dat`. Первый компонент пути (`dir`) будет обработан как псевдоним директории. Если псевдоним не будет найден в `directories.conf`, то путь будет обработан как относительный (относительно директории, в которой размещен основной файл базы данных). Все директории, используемые в пути, должны быть созданы заранее. В системную таблицу `RDB$TABLESPACES` в любом случае будет записан абсолютный путь к файлу табличного пространства.

См. также операторы ALTER TABLESPACE, DROP TABLESPACE.

CREATE TRIGGER

Оператор CREATE TRIGGER создает новый триггер. Синтаксис:

Листинг Ж.54. Синтаксис оператора CREATE TRIGGER

```
CREATE TRIGGER <имя триггера> {
      <объявление табличного триггера>
    | <объявление табличного триггера в стандарте SQL-2003>
    | <объявление триггера базы данных>
    | <объявление DDL триггера> }
[SQL SECURITY {DEFINER | INVOKER}]
{ EXTERNAL NAME '<внешний модуль>' ENGINE <имя движка> [AS <тело внеш. триг.>]} |
{
   AS
      [<объявление> [<объявление> ...] ]
   BEGIN
      <блок операторов>
   END }

<объявление табличного триггера> ::=
         FOR {<имя таблицы> | <имя представления>}
         [ACTIVE | INACTIVE]
         {BEFORE | AFTER} <список событий таблицы (представления)>
         [POSITION <порядок срабатывания триггера>]

<объявление табличного триггера в стандарте SQL-2003> ::=
         [ACTIVE | INACTIVE]
         {BEFORE | AFTER} <список событий таблицы (представления)>
         [POSITION <порядок срабатывания триггера>]
         ON {<имя таблицы> | <имя представления>}

<объявление триггера базы данных> ::=
         [ACTIVE | INACTIVE]
         ON <событие соединения или транзакции>
         [POSITION <порядок срабатывания триггера>]

<объявление DDL триггера> ::=
         [ACTIVE | INACTIVE]
         {BEFORE | AFTER} <список DDL событий>
         [POSITION <порядок срабатывания триггера>]

<список событий таблицы (представления)> ::= <событие DML> [OR <событие DML>...]

<событие DML> ::= { INSERT | UPDATE | DELETE }

<событие соединения или транзакции> ::= {
           CONNECT
         | DISCONNECT
         | TRANSACTION START
         | TRANSACTION COMMIT
         | TRANSACTION ROLLBACK }
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
<список DDL событий> ::= {
            ANY DDL STATEMENT
          | <DDL событие> [OR <DDL событие> ...] }

<DDL событие> ::=
            CREATE|ALTER|DROP TABLE
          | CREATE|ALTER|DROP PROCEDURE
          | CREATE|ALTER|DROP FUNCTION
          | CREATE|ALTER|DROP TRIGGER
          | CREATE|ALTER|DROP EXCEPTION
          | CREATE|ALTER|DROP VIEW
          | CREATE|ALTER|DROP DOMAIN
          | CREATE|ALTER|DROP ROLE
          | CREATE|ALTER|DROP SEQUENCE
          | CREATE|ALTER|DROP USER
          | CREATE|ALTER|DROP INDEX
          | CREATE|DROP COLLATION
          | ALTER CHARACTER SET
          | CREATE|ALTER|DROP PACKAGE
          | CREATE|DROP PACKAGE BODY
          | CREATE|ALTER|DROP MAPPING

<объявление> ::= <объявление локальной переменной>;
                 | <объявление курсора>;
                 | <объявление процедуры/функции>
                 | <реализация процедуры/функции>
```

Табличный триггер может быть создан владельцем таблицы (представления), для которого создается DML триггер, администратором и пользователем с привилегией `ALTER ANY {TABLE|VIEW\`. Триггеры на события базы данных и на события изменения метаданных может создаваться только владельцем базы данных, администратором и пользователем с привилегией `ALTER DATABASE`.

С помощью этого оператора можно:

- задать имя триггера (может содержать до 63 символов и должно быть уникальным среди имен всех триггеров базы данных);
- объявить табличный триггер, триггер на события базы данных или DML триггер;
- указать состояние триггера: активный или неактивный;
- указать место расположения триггера во внешнем модуле
- указать в контексте какого пользователя будет выполняться триггер;
- описать произвольное количество локальных переменных, именованных курсоров и подпрограммы (подпроцедуры и подфункции);
- задать блок операторов в теле триггера.

Подробно оператор описан в главе 20.

См. также операторы ALTER TRIGGER, CREATE OR ALTER TRIGGER, RECREATE TRIGGER, DROP TRIGGER.

CREATE USER

Для создания новой учетной записи пользователя используется следующий синтаксис:

Листинг Ж.55. Синтаксис оператора CREATE USER

```
CREATE USER <логин> PASSWORD '<пароль>'
   [FIRSTNAME '<имя пользователя>']
   [MIDDLENAME '<отчество пользователя>']
   [LASTNAME '<фамилия пользователя>']
   [ACTIVE | INACTIVE]
   [USING PLUGIN 'имя плагина']
   [TAGS (<атрибут> [, <атрибут> ...] )]
   [GRANT ADMIN ROLE]

<атрибут> ::= <имя атрибута> = 'строковое значение'
```

Пользователь должен отсутствовать в текущей базе данных безопасности РЕД Базе Данных иначе будет выдано соответствующее сообщение об ошибке.

Начиная с версии 3.0 имена пользователей подчиняются общему правилу наименования идентификаторов объектов метаданных. Таким образом, пользователь с именем `"Alex"` и с именем `"ALEX"` будут разными пользователями.

Предложение `PASSWORD` задаёт пароль пользователя. Максимальная длина пароля зависит от того какой менеджер пользователей задействован (параметр `UserManager` в файле конфигурации `firebird.conf`). Для менеджер пользователей `SRP` эффективная длина пароля ограничена 20 байтами. Для менеджер пользователей `Legacy_UserManager` максимальная длина пароля равна 8 байт.

Необязательные предложения `FIRSTNAME, MIDDLENAME` и `LASTNAME` задают дополнительные атрибуты пользователя, такие как имя пользователя, отчество и фамилия соответственно.

Кроме того можно задать неограниченное количество пользовательских атрибутов с помощью необязательного предложения `TAGS`.

Если при создании учётной записи будет указан атрибут `INACTIVE`, то пользователь будет создан в "неактивном состоянии", т.е. подключиться с его учётной записью будет невозможно. При указании атрибута `ACTIVE` пользователь будет создан в активном состоянии (по умолчанию).

С опцией `GRANT ADMIN ROLE` создаётся новый пользователь с правами роли `RDB$ADMIN` в базе данных пользователя (`security5.fdb`). Это позволяет ему управлять учётными записями пользователей, но не дает ему специальных полномочий в обычных базах данных.

Необязательное предложение `USING PLUGIN` позволяет явно указывать какой плагин управления пользователями будет использован. По умолчанию используется тот плагин, который был указан первым в списке параметра `UserManager` в файле конфигурации `firebird.conf`. Допустимыми являются только значения, перечисленные в параметре `UserManager`. Следует учитывать, что одноименные пользователи, созданные с помощью разных плагинов управления пользователями — это разные пользователи.

Если предложение `USING PLUGIN` не указано, то при добавлении пользователя он сам добавляется во все плагины из списка параметра `DefaultUserManagers` (в том числе его атрибуты).

```
CREATE USER jon PASSWORD '87654321'
FIRSTNAME 'Jon'
LASTNAME 'Snow'
TAGS (BIRTHYEAR = '283' , CITY = 'Winterfell');
```

См. также операторы ALTER USER, DROP USER.

CREATE VIEW

Оператор CREATE VIEW создает новое представление. Его синтаксис:

Листинг Ж.56. Синтаксис оператора CREATE VIEW

```
CREATE VIEW <имя представления> [<список столбцов>]
AS <оператор SELECT>
[WITH CHECK OPTION]

<список столбцов> ::= (столбец [, столбец ...])
```

Представления могут создаваться администраторами и пользователями с привилегией CREATE VIEW. Пользователь, создавший представление, становится его владельцем. Для создания представления пользователями, которые не имеют административных привилегий, необходимы также привилегии на чтение (SELECT) данных из базовых таблиц и представлений, и привилегии на выполнение (EXECUTE) используемых селективных хранимых процедур. Для разрешения вставки, обновления и удаления через представление, необходимо чтобы создатель (владелец) представления имел привилегии INSERT, UPDATE и DELETE на базовые объекты метаданных.

- Имя представления должно быть уникальным среди имен всех представлений, таблиц и хранимых процедур базы данных.
- Оператор SELECT может быть оператором выборки данных любой сложности. Здесь можно выполнять объединение (UNION) и соединение (JOIN) различных таблиц, использовать предложение WHERE для задания условий выбора строк.
- Представление может быть изменяемым (в двух вариантах — естественно изменяемым или изменяемым при помощи вспомогательных триггеров) или неизменяемым, только для чтения (read-only).
- Необязательное предложение WITH CHECK OPTION задает для изменяемого представления требование проверки соответствия вновь вводимых или изменяемых данных условию, заданному в предложении WHERE.

Подробно оператор создания представления описан в главе 15.

Возможности оператора SELECT см. в главе 11.

См. также операторы CREATE OR ALTER VIEW, RECREATE VIEW, DROP VIEW, ALTER VIEW, INSERT, UPDATE, UPDATE OR INSERT, DELETE.

DECLARE EXTERNAL FUNCTION

Оператор DECLARE EXTERNAL FUNCTION объявляет функцию, определенную пользователем (UDF — User Defined Function). Его синтаксис:

Листинг Ж.57. Синтаксис оператора DECLARE EXTERNAL FUNCTION

```
DECLARE EXTERNAL FUNCTION <имя UDF>
     [<тип данных> [{BY DESCRIPTOR} | NULL] | CSTRING (<целое>) [NULL]
     [, <тип данных> [{BY DESCRIPTOR} | NULL] | CSTRING (<целое>) [NULL] ...]]
RETURNS {
       <тип данных> [BY VALUE | BY DESCRIPTOR]
     | CSTRING (<целое>)
     | PARAMETER <номер> }
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
[FREE_IT]
ENTRY_POINT '<имя точки входа>'
MODULE_NAME '<имя модуля>';
```

Объявить внешнюю функцию может пользователь с административными привилегиями и пользователь с привилегией CREATE FUNCTION. Пользователь, объявивший внешнюю функцию, становится её владельцем.

С помощью данного оператора можно:

- задать имя функции;
- перечислить входные параметры, передаваемые функции
- задать возвращаемый тип данных;
- указать как возвращается данное — по значению или по ссылке;
- задать передачу параметров по дескриптору;
- задать имя модуля и имя точки входа для функции в модуле.

Подробнее оператор рассмотрен в :Приложение Е.

См. также оператор ALTER EXTERNAL FUNCTION, DROP EXTERNAL FUNCTION.

DECLARE FILTER

Оператор DECLARE FILTER объявляет существующий BLOB фильтр в базе данных. BLOB фильтр – объект базы данных, подпрограмма, которая транслирует объекты BLOB из одного формата в другой. Форматы объектов BLOB задаются с помощью подтипов BLOB. Внешние функции для преобразования BLOB типов хранятся в динамических библиотеках и загружаются по необходимости.

Листинг Ж.58. Синтаксис оператора DECLARE FILTER

```
DECLARE FILTER <имя фильтра>
INPUT_TYPE <подтип BLOB> OUTPUT_TYPE <подтип BLOB>
ENTRY_POINT 'Имя экспортируемой функции'
MODULE_NAME 'Имя модуля с фильтром'

<подтип BLOB> ::= номер подтипа | <мнемоника подтипа>

<мнемоника подтипа> ::= binary | text | blr | acl | ranges | summary |
                        format | transaction_description |
                        external_file_description | <user_defined>
```

Создать BLOB фильтр может администратор и пользователь с привилегией CREATE FILTER.

С помощью данного оператора можно:

- задать имя BLOB фильтра (должно быть уникальным среди имен BLOB фильтров);
- установить подтип BLOB преобразуемого объекта;
- установить подтип BLOB создаваемого объекта;
- указать имя модуля и имя экспортируемой функции (точки входа) в модуле.

Подробнее оператор рассмотрен в главе 13.

См. также оператор DROP FILTER.

DELETE

Оператор DELETE используется для удаления существующих строк из таблицы или представления. Он позволяет удалить все или группу строк таблицы (таблиц представления). Синтаксис оператора:

Листинг Ж.59. Синтаксис оператора DELETE

```
DELETE FROM {<имя таблицы> | <имя представления>} [[AS] <псевдоним>]
[WHERE { <условие поиска> | CURRENT OF <имя курсора>}]
[PLAN <план>]
[ORDER BY <упорядочиваемый элемент> [, <упорядочиваемый элемент> ... ]]
[ROWS <m> [TO <n>]]
[SKIP LOCKED]
[RETURNING {<список возвращаемых значений> | *} [INTO <переменные>]]

<список возвращаемых значений> ::= <имя столбца> [[AS] <алиас>] [, <имя столбца> [[AS]
<алиас>]...]

<переменные> ::= [:]<имя переменной> [, [:]<имя переменной> ...];
```

Удалять данные из таблицы или представления может их владелец, пользователь SYSDBA, пользователь операционной системы root (Linux), trusted user (Windows), а также пользователь, которому предоставлено право на удаление строк из таблицы или представления оператором GRANT DELETE — см. документ «Руководство администратора». Если удаление строки таблицы влечет и удаление подчиненных строк из дочерней таблицы, то и к этой таблице пользователь должен иметь соответствующие полномочия.

- Предложение WHERE ограничивает набор удаляемых записей заданным условием или текущей строкой именованного курсора;
- Ключевое слово PLAN задает план выборки данных для удаления;
- Предложение ORDER BY используется для упорядочения результатов выборки;
- Предложение SKIP LOCKED позволяет пропускать записи, заблокированные другими транзакциями, вместо того чтобы ждать их разблокирования или выдавать ошибку конфликта обновления;
- Предложение RETURNING позволяет вернуть вызвавшей программе значения указанных столбцов удаленной строки.

Подробнее об операторе удаления см. в главе 11.

См. также операторы INSERT, UPDATE, UPDATE OR INSERT.

DROP COLLATION

Оператор DROP COLLATION удаляет указанную сортировку. Сортировка должна присутствовать в базе данных, иначе будет выдана соответствующая ошибка.

Листинг Ж.60. Синтаксис оператора DROP COLLATION

```
DROP COLLATION <имя сортировки>;
```

Выполнить данный оператор может только администратор, владелец сортировки и пользователь с привилегией DROP ANY COLLATION.

См. также оператор CREATE COLLATION.

DROP DATABASE

Для удаления существующей базы данных используется оператор SQL `DROP DATABASE`. Его синтаксис:

Листинг Ж.61. Синтаксис оператора `DROP DATABASE`

```
DROP DATABASE;
```

Прежде чем удалять базу данных, с ней нужно соединиться. Оператор удаляет первичный, все вторичные файлы базы данных и все файлы оперативных копий (см. CREATE SHADOW), связанные с этой базой данных.

Удалять базу данных может ее владелец, администратор и пользователь с привилегией `DROP DATABASE`.

Подробно оператор описан в главе 6.

См. также операторы CREATE DATABASE, ALTER DATABASE, CREATE SHADOW, DROP SHADOW.

DROP DOMAIN

Оператор `DROP DOMAIN` удаляет из базы данных существующий домен. Синтаксис оператора:

Листинг Ж.62. Синтаксис оператора `DROP DOMAIN`

```
DROP DOMAIN <имя домена>;
```

Нельзя удалить домен, на который ссылаются столбцы таблиц базы данных. Предварительно нужно удалить все столбцы, ссылающиеся на этот домен.

Удалить существующий домен может владелец домена (его создатель), пользователь с административными привилегиями или пользователь с привилегией `DROP ANY DOMAIN`.

См. также операторы CREATE DOMAIN, ALTER DOMAIN.

DROP EXCEPTION

Оператор позволяет удалить указанное пользовательское исключение из базы данных. Его синтаксис:

Листинг Ж.63. Синтаксис оператора `DROP EXCEPTION`

```
DROP EXCEPTION <имя исключения>;
```

Удалить пользовательское исключение может администратор, владелец исключения или пользователь с привилегией `DROP ANY EXCEPTION`.

При наличии зависимостей для существующего исключения удаления не будет выполнено.

См. также операторы CREATE EXCEPTION, ALTER EXCEPTION, операторы PSQL WHEN-DO, EXCEPTION.

DROP EXTERNAL FUNCTION

Оператор DROP EXTERNAL FUNCTION удаляет объявление функции определённой пользователем из базы данных. Если есть зависимости от внешней функции, то удаления не произойдёт и будет выдана соответствующая ошибка.

Листинг Ж.64. Синтаксис оператора DROP EXTERNAL FUNCTION

```
DROP EXTERNAL FUNCTION <имя UDF>;
```

Удалить внешнюю функцию может администратор, владелец функции (ее создатель) или пользователь с привилегией DROP ANY FUNCTION.

См. также операторы DECLARE EXTERNAL FUNCTION, ALTER EXTERNAL FUNCTION.

DROP FILTER

Данный оператор удаляет объявление BLOB фильтра из базы данных.

Листинг Ж.65. Синтаксис оператора DROP FILTER

```
DROP FILTER <имя фильтра>
```

Удаление BLOB фильтра из базы данных делает его недоступным из базы данных. Но динамическая библиотека, в которой расположена функция преобразования, остается нетронутой.

Удалить объявление BLOB фильтра администратор, владелец фильтра и пользователь с привилегией DROP ANY FILTER.

См. также оператор DECLARE FILTER.

DROP FUNCTION

Для удаления существующей хранимой функции используется оператор DROP FUNCTION. Синтаксис оператора:

Листинг Ж.66. Синтаксис оператора удаления хранимой функции DROP FUNCTION

```
DROP FUNCTION <имя хранимой процедуры>
```

Если от хранимой функции существуют зависимости, то при попытке удаления такой функции будет выдана соответствующая ошибка.

Удалить хранимую функцию может администратор, владелец хранимой функции и пользователь с привилегией DROP ANY FUNCTION.

См. также операторы CREATE FUNCTION, RECREATE FUNCTION, ALTER FUNCTION, CREATE OR ALTER FUNCTION.

DROP GENERATOR

Оператор DROP GENERATOR (DROP SEQUENCE) удаляет генератор из базы данных. Синтаксис:

Листинг Ж.67. Синтаксис оператора DROP GENERATOR

```
DROP {GENERATOR | SEQUENCE} <имя генератора>;
```

Нельзя удалить генератор, если на него есть ссылки в триггерах или хранимых процедурах базы данных.

Удалять генераторы могут администраторы, владельцы последовательности и пользователи с привилегией `DROP ANY SEQUENCE` (`DROP ANY GENERATOR`).

Подробно работа с генераторами описана в главе 12.

См. также операторы DROP SEQUENCE, CREATE GENERATOR, CREATE SEQUENCE, SET GENERATOR, ALTER SEQUENCE, функцию GEN_ID, конструкцию NEXT VALUE FOR.

DROP INDEX

Оператор `DROP INDEX` удаляет существующий индекс из базы данных. Его синтаксис:

Листинг Ж.68. Синтаксис оператора `DROP INDEX`

```
DROP INDEX <имя индекса>;
```

Нельзя удалить индекс, созданный автоматически системой для первичного, уникального или внешнего ключа.

Удалить индекс может только владелец таблицы, для которой создан индекс, администратор и пользователь с привилегией `ALTER ANY TABLE`.

Подробно оператор описан в главе 14.

См. также операторы CREATE INDEX, ALTER INDEX.

DROP JOB

Оператор удаления задания доступен создателю задания и `SYSDBA`.

Листинг Ж.69. Синтаксис оператора `DROP JOB`

```
DROP JOB <имя_задания>;
```

Подробно оператор описан в главе 23.

См. также операторы CREATE JOB, ALTER JOB.

DROP MAPPING

Оператор `DROP MAPPING` удаляет существующее отображение. Если указана опция `GLOBAL`, то будет удалено глобальное отображение.

Листинг Ж.70. Синтаксис оператора `DROP MAPPING`

```
DROP [GLOBAL] MAPPING <имя отображения>
```

Удалить отображение может `SYSDBA`, владелец базы данных (если отображение локальное), пользователь с ролью `RDB$ADMIN`, пользователь `root` (Linux).

См. также операторы CREATE OR ALTER MAPPING, CREATE MAPPING, ALTER MAPPING.

DROP PACKAGE

Оператор DROP PACKAGE удаляет существующий заголовок пакета. Синтаксис оператора:

Листинг Ж.71. Синтаксис оператора удаления заголовка пакета DROP PACKAGE

```
DROP PACKAGE <имя пакета>
```

Выполнить удаление заголовка пакета может администратор, владелец пакета или пользователь с привилегией DROP ANY PACKAGE.

Перед удалением заголовка пакета, необходимо выполнить удаление тела пакета (DROP PACKAGE BODY), иначе будет выдана ошибка. Если от заголовка пакета существуют зависимости, то при попытке удаления такого заголовка будет выдана соответствующая ошибка.

Подробно оператор описан в главе 21.

См. также операторы DROP PACKAGE BODY, RECREATE PACKAGE, CREATE PACKAGE, ALTER PACKAGE, CREATE OR ALTER PACKAGE.

DROP PACKAGE BODY

Оператор DROP PACKAGE BODY удаляет существующее тело пакета. Синтаксис оператора:

Листинг Ж.72. Синтаксис оператора удаления тела пакета DROP PACKAGE BODY

```
DROP PACKAGE BODY <имя пакета>
```

Выполнить удаление заголовка пакета может администратор, владелец пакета или пользователь с привилегией DROP ANY PACKAGE.

Подробно оператор описан в главе 21.

См. также операторы DROP PACKAGE, RECREATE PACKAGE BODY, CREATE PACKAGE BODY.

DROP POLICY

Для удаления политики безопасности администратору необходимо соединиться с какой-либо базой данных. Для удаления политики используется оператор DROP POLICY. Синтаксис этого оператора приведен ниже:

```
DROP POLICY <имя политики>
```

См. также операторы CREATE POLICY, ALTER POLICY.

DROP PROCEDURE

Для удаления существующей хранимой процедуры используется оператор DROP PROCEDURE. Синтаксис оператора:

Листинг Ж.73. Синтаксис оператора DROP PROCEDURE

```
DROP PROCEDURE <имя хранимой процедуры>;
```

Нельзя удалить хранимую процедуру, к которой существуют обращения из других хранимых процедур, триггеров и представлений. Также нельзя удалить хранимую процедуру, которая выполняется в настоящий момент.

Удалить хранимую процедуру может ее создатель и администратор (пользователь с ролью RDB$ADMIN) и пользователь с привилегией DROP ANY PROCEDURE.

Подробно оператор описан в главе 18.

См. также операторы ALTER PROCEDURE, CREATE PROCEDURE, CREATE OR ALTER PROCEDURE, RECREATE PROCEDURE, EXECUTE PROCEDURE.

DROP ROLE

Оператор удаляет существующую роль из базы данных. Синтаксис оператора:

Листинг Ж.74. Синтаксис оператора DROP ROLE

```
DROP ROLE <имя роли>;
```

Удаляется созданная ранее роль. При этом также удаляются все привилегии пользователей, полученные с этой ролью.

Роль может удалить ее владелец или администратор (пользователь с ролью RDB$ADMIN).

См. также операторы CREATE ROLE, GRANT, REVOKE, CONNECT.

DROP SEQUENCE

Другое название для оператора удаления генератора.

Листинг Ж.75. Синтаксис оператора DROP SEQUENCE

```
DROP SEQUENCE <имя генератора>;
```

Удалять генераторы могут администраторы, владельцы последовательности и пользователи с привилегией DROP ANY SEQUENCE (DROP ANY GENERATOR).

Оператор удаляет указанный генератор. Подробно работа с генераторами описана в главе 12.

См. также оператор DROP GENERATOR.

DROP SHADOW

Оператор DROP SHADOW удаляет указанную оперативную копии из базы данных, с которой в настоящий момент существует соединение. Его синтаксис:

Листинг Ж.76. Синтаксис оператора DROP SHADOW

```
DROP SHADOW <номер оперативной копии> [{PRESERVE | DELETE} FILE];
```

Номер оперативной копии — положительное число, идентифицирующее набор файлов ранее созданной оперативной копии.

При удалении оперативной копии прекращается процесс дублирования данных в этой оперативной копии. Если указана опция DELETE FILE (по умолчанию), то будут также удалены и все связанные файлы с этой теневой копией. Если указана опция PRESERVE FILE, то файлы останутся не тронутыми. Это может быть полезно, если делать резервную копию с теневого файла.

Оперативная копия может быть удалена владельцем базы данных, администратором пользователем с привилегией ALTER DATABASE.

Подробно оператор описан в главе 7.

См. также операторы CREATE DATABASE, ALTER DATABASE, DROP DATABASE, CREATE SHADOW.

DROP TABLE

Для удаления существующей в базе данных таблицы используется оператор DROP TABLE. Удалять таблицу может ее владелец, администратор и пользователь с привилегией DROP ANY TABLE.

Синтаксис оператора:

Листинг Ж.77. Синтаксис оператора DROP TABLE

```
DROP TABLE <имя таблицы>;
```

Нельзя удалить таблицу, которая является родительской в связке внешний ключ/первичный (уникальный) ключ. Нельзя удалить таблицу, на которую существуют ссылки в триггерах (за исключением триггеров, написанных пользователем именно для этой таблицы), которая используется в хранимой процедуре или в представлении.

Подробно оператор описан в главе 9.

См. также операторы CREATE TABLE, ALTER TABLE.

DROP TABLESPACE

Для удаления существующего табличного пространства используется оператор DROP TABLESPACE.

Листинг Ж.78. Синтаксис оператора удаления табличного пространства DROP TABLESPACE

```
DROP TABLESPACE <имя табличного пространства>;
```

Если в удаляемом табличном пространстве хранятся таблицы или индексы, то будет выдано сообщение об ошибке.

Права на удаление табличных пространств есть только у администраторов и пользователей с привилегией DROP ANY TABLESPACE.

См. также операторы CREATE TABLESPACE, ALTER TABLESPACE.

DROP TRIGGER

Оператор DROP TRIGGER удаляет существующий триггер. Синтаксис оператора:

Листинг Ж.79. Синтаксис оператора DROP TRIGGER

```
DROP TRIGGER <имя триггера>;
```

DML триггер может быть удален администратором и владельцем таблицы или представления или пользователем с привилегией ALTER ANY {TABLE | VIEW\. Триггеры для событий базы данных и триггеры событий на изменение метаданных может удалить администратор, владелец базы данных или пользователь с привилегией ALTER DATABASE.

Нельзя удалить триггер, автоматически созданный системой для поддержания ограничений PRIMARY KEY, CHECK и FOREIGN KEY. Остальные триггеры не имеют никаких зависимостей, которые ограничили бы возможности удаления триггеров.

Подробно оператор описан в главе 20.

См. также операторы CREATE TRIGGER, CREATE OR ALTER TRIGGER, RECREATE TRIGGER, ALTER TRIGGER.

DROP USER

Можно управлять учётными записями пользователей средствами операторов SQL. Для удаления существующей учетной записи пользователя используется следующий синтаксис:

Листинг Ж.80. Синтаксис оператора DROP USER

```
DROP USER <логин>
[USING PLUGIN 'имя плагина'];
```

Для удаления учетной записи пользователя текущий пользователь должен обладать административными привилегиями в базе данных безопасности.

Необязательное предложение USING PLUGIN позволяет явно указывать какой плагин управления пользователями будет использован. По умолчанию используется тот плагин, который был указан первым в списке параметра UserManager в файле конфигурации firebird.conf. Допустимыми являются только значения, перечисленные в параметре UserManager. Следует учитывать, что одноименные пользователи, созданные с помощью разных плагинов управления пользователями — это разные пользователи.

Если предложение USING PLUGIN не указано, то при удалении пользователя он сам удаляется из всех плагинов из списка параметра DefaultUserManagers.

См. также операторы CREATE USER, ALTER USER, CREATE OR ALTER USER.

DROP VIEW

Для удаления существующего в базе данных представления используется оператор DROP VIEW. Его синтаксис:

Листинг Ж.81. Синтаксис оператора DROP VIEW

```
DROP VIEW <имя представления>
```

Представление нельзя удалить, если не него есть ссылки в другом представлении, в хранимой процедуре или в ограничении CHECK столбца таблицы или соответствующего ограничения таблицы.

Удалить представление может только владелец представления, администратор, пользователь с привилегией DROP ANY VIEW.

В SQL не существует средств для изменения созданных оператором CREATE VIEW представлений. Для того чтобы изменить представление, его нужно удалить, а затем создать с требуемыми новыми характеристиками.

Подробно оператор описан в главе 15.

См. также операторы CREATE VIEW, RECREATE VIEW, CREATE OR ALTER VIEW, ALTER VIEW.

EXECUTE BLOCK

Оператор EXECUTE BLOCK позволяет из декларативной части SQL (DSQL) выполнять действия, доступные в хранимых процедурах и триггерах.

Листинг Ж.82. Синтаксис оператора EXECUTE BLOCK

```
EXECUTE BLOCK
   [(<список входных параметров>)]
   [RETURNS (<список выходных параметров>)]
AS
   [<объявление> [<объявление> ...] ]
BEGIN
   <блок операторов>
END;

<список входных параметров>::=<описание параметра>=? [,<описание параметра>=?...]

<список выходных параметров> ::= <описание параметра> [, <описание параметра>]

<описание параметра> ::= <имя параметра> <тип> [NOT NULL]
                         [COLLATE <порядок сортировки>]

<тип> ::=  <тип данных SQL>
         | [TYPE OF] <имя домена>
         | TYPE OF COLUMN <имя таблицы/представления>.<имя столбца>

<объявление> ::=  объявление локальной переменной
                | объявление курсора
                | <объявление подпрограммы (процедуры или функции)>
                | <реализация подпрограммы (процедуры или функции)>
```

Список входных параметров можно задавать в случае, если оператор EXECUTE BLOCK выполняется из какой-либо графической программы, которая имеет возможность задать значения входных параметров. В isql такой список использовать нельзя.

Тело анонимного PSQL блока может содержать объявление локальных переменных, курсоров и блок PSQL операторов. В блоке операторов выполняются произвольные действия по обработке данных.

Анонимный PSQL блок не определяется и сохраняется как объект метаданных.

Подробно оператор описан в главе 11.

См. также операторы CREATE PROCEDURE, ALTER PROCEDURE, DROP PROCEDURE.

EXECUTE PROCEDURE

В DSQL, в языке хранимых процедур и триггеров и при использовании утилиты isql можно вызвать выполняемую хранимую процедуру, используя оператор EXECUTE PROCEDURE. Его синтаксис:

Листинг Ж.83. Синтаксис оператора EXECUTE PROCEDURE

```
EXECUTE PROCEDURE <имя процедуры> [(<параметр> [, <параметр>] ...)]
[RETURNING_VALUES (<параметр> [, <параметр>] ...)];
```

При вызове хранимой процедуры можно после имени процедуры указать список входных параметров для процедуры. Если процедура получает параметры, то список входных параметров в опера-

торе EXECUTE PROCEDURE является обязательным. При этом требуется полное соответствие количества параметров и их типов данных.

Если этот оператор вызывается из isql, то нельзя использовать предложение RETURNING_VALUES.

Подробно оператор описан в главе 11.

См. также операторы CREATE PROCEDURE, ALTER PROCEDURE, DROP PROCEDURE, SELECT.

GRANT

Оператор предоставляет одну или несколько привилегий для объектов базы данных пользователям, ролям, хранимым процедурам, функциям, пакетам, триггерам и представлениям. Синтаксис оператора:

Листинг Ж.84. Синтаксис оператора GRANT

```
GRANT <табличные привилегии> ON [TABLE] {<имя таблицы/представления>}
   TO <список получателей привилегий> [WITH GRANT OPTION]
   [{GRANTED BY|AS} [USER] <имя грантора>]

GRANT EXECUTE ON {PROCEDURE | FUNCTION | PACKAGE} <имя процедуры/функции/пакета>
   TO <список получателей привилегий> [WITH GRANT OPTION]
   [{GRANTED BY|AS} [USER] <имя грантора>]

GRANT USAGE ON {EXCEPTION <имя искл-я>|{GENERATOR | SEQUENCE} <имя генератора>}
   TO <список получателей привилегий> [WITH GRANT OPTION]
   [{GRANTED BY|AS} [USER] <имя грантора>]

GRANT {ALL [PRIVILEGES] | {CREATE|ALTER ANY|DROP ANY} [, {CREATE|ALTER ANY| DROP ANY}.
..] } <объект>
   TO <список получателей привилегий> [WITH GRANT OPTION]
   [{GRANTED BY|AS} [USER] <имя грантора>]

GRANT CREATE DATABASE TO {USER <имя пользователя>|ROLE <имя роли>|GROUP <имя группы в
Unix>} [,{USER <имя пользователя>|ROLE <имя роли>|GROUP <имя группы в Unix>}...]

GRANT {ALL [PRIVILEGES] | {ALTER|DROP} [,{ALTER|DROP}...]} DATABASE
   TO <список получателей привилегий> [WITH GRANT OPTION]
   [{GRANTED BY|AS} [USER] <имя грантора>]

GRANT [DEFAULT] <имя роли> [, [DEFAULT] <имя роли> ...]
   TO [USER]|[ROLE] <имя польз-я/роли> [, [USER]|[ROLE] <имя польз-я/роли>...]
   [WITH ADMIN OPTION] [{GRANTED BY | AS} [USER] <имя грантора>]

GRANT POLICY <имя_политики> TO <имя_пользователя>

<табличные привилегии> ::= ALL [PRIVILEGES] | <привилегия> [, <привилегия>...]

<привилегия> ::=   SELECT
                 | DELETE
                 | INSERT
                 | UPDATE [(<имя столбца [, <имя столбца>...]>)]
                 | REFERENCES [(<имя столбца [, <имя столбца>...]>)]
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
<объект> ::=  TABLE
            | TABLESPACE
            | VIEW
            | PROCEDURE
            | FUNCTION
            | PACKAGE
            | GENERATOR
            | SEQUENCE
            | DOMAIN
            | EXCEPTION
            | ROLE
            | CHARACTER SET
            | COLLATION
            | FILTER
            | JOB

<список получателей привилегий> ::= {<объект получатель>|<пользователь получатель>}
                                  [, {<объект получатель>|<пользователь получатель>}..
.]

<объект получатель> ::=  PROCEDURE <имя процедуры>
                       | FUNCTION <имя функции>
                       | PACKAGE <имя пакета>
                       | TRIGGER <имя триггера>
                       | VIEW <имя представления>
                       | SYSTEM PRIVILEGE <системная привилегия> }

<пользователь получатель> ::=  [USER] <имя пользователя>
                             | [ROLE] <имя роли>
                             | GROUP <имя группы в Unix>
```

Все привилегии по доступу к объектам базы данных хранятся в самой базе, и не могут быть применены к любой другой базе данных.

Авторизованный пользователь не имеет никаких привилегий до тех пор, пока какие либо права не будут предоставлены ему явно. SYSDBA или владелец объекта могут выдавать привилегии другим пользователям, в том числе и привилегии на право выдачи привилегий другим пользователям.

Оператор позволяет выполнить одно из следующих действий:

- предоставить одну или несколько привилегий для таблиц и представлений пользователям, ролям, представлениям, хранимым процедурам, триггерам, пакетам и функциям;
- выдать права на выполнение процедуры, функции или пакета пользователям, ролям, представлениям, хранимым процедурам, триггерам, пакетам и функциям;
- контролирует доступ к исключениям, последовательностям и генераторам;
- предоставить одну или несколько привилегий на выполнение DDL операций над основными объектами базы данных пользователям, ролям, представлениям, хранимым процедурам, триггерам, пакетам и функциям;
- назначает привилегии на создание, удаление и изменение базы данных пользователям, ролям, представлениям, хранимым процедурам, триггерам, пакетам и функциям;
- назначить указанные роли группе перечисленных пользователей;
- назначить политику конкретному пользователю.

Подробно оператор описан в главе 26.

См. также операторы CREATE ROLE, DROP ROLE, REVOKE, CONNECT.

INSERT

Оператор INSERT добавляет строки в обычную таблицу или в таблицу изменяемого представления. Синтаксис оператора:

Листинг Ж.85. Синтаксис оператора INSERT

```
INSERT INTO {<имя таблицы> | <имя представления>}
[OVERRIDE {SYSTEM | USER} VALUE]
{ DEFAULT VALUES | [(<список столбцов>)] <источник значений>}
[RETURNING { <список возвращаемых значений> | *} [INTO <переменные>]]

<источник значений> ::=  VALUES (<значение>| DEFAULT [,<значение>| DEFAULT...])
                       | <поиск многих>

<список столбцов> ::= <имя столбца> [, <имя столбца> ...]

<список возвращаемых значений> ::= <имя столбца> [[AS] <алиас>] [, <имя столбца>
[[AS] <алиас>]...]

<переменные> ::= [:]<имя переменной> [, [:]<имя переменной> ...]

<значение> ::=  <литерал>
              | <выражение>
              | <встроенная функция>
              | <UDF> [(<параметр> [, <параметр> ...])]
              | <обращение к хранимой процедуре> [(<параметр> [, <параметр>]...)]
              | NEXT VALUE FOR <имя генератора>
              | (<выбор одного>) }
```

Добавлять данные в таблицу может ее владелец, администратор базы данных, а также пользователь, которому предоставлено право добавлять данные в таблицу (в таблицы, базовые для используемого представления) оператором GRANT INSERT.

Оператор позволяет выполнить одно из следующих действий:

- добавить строку, для которой указаны конкретные значения столбцов;
- добавить строку, которая будет во всех столбцах содержать значения по умолчанию (предложение DEFAULT VALUES).

Подробнее об операторе добавления см. в главе 11.

См. также операторы UPDATE, UPDATE OR INSERT, DELETE, SELECT, SET TRANSACTION, функции AVG, SUM, MIN, MAX, COUNT, CAST, GEN_ID, NEXT VALUE FOR.

MERGE

Оператор MERGE объединяет данные в таблицу или представление.

Листинг Ж.86. Синтаксис оператора MERGE

```
MERGE INTO <имя таблицы | имя представления> [[AS] <алиас>]
USING <таблица|представление|хранимая процедура|производная таблица> [AS <алиас>]
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
ON <условие соединения>
<предложение WHEN> [<предложение WHEN> ...]
[PLAN <выражение для построения плана>]
     [ORDER BY <выражение для упорядочивания выборки>]
[RETURNING <список возвращаемых выражений>|[{OLD. | NEW.}]*
   [INTO <список переменных>]]

<предложение WHEN> ::= <предложение WHEN MATCHED>
                         | <предложение WHEN NOT MATCHED BY TARGET>
                         | <предложение WHEN NOT MATCHED BY SOURCE>

<предложение WHEN MATCHED> ::=
       WHEN MATCHED [ AND <доп.условие> ]
       THEN {UPDATE SET <список назначений> | DELETE}

<предложение WHEN NOT MATCHED BY TARGET> ::=
       WHEN NOT MATCHED [ BY TARGET ] [ AND <доп.условие> ]
       THEN INSERT [(<столбцы>)] VALUES (<значения>)

<предложение WHEN NOT MATCHED BY SOURCE> ::=
       WHEN NOT MATCHED BY SOURCE [ AND <доп.условие> ]
       THEN { UPDATE SET <список назначений> | DELETE }

<список назначений> ::= <имя столбца>={<значение>|DEFAULT} [, <имя столбца>= {
<значение>|DEFAULT} ...]

<столбцы> ::= <имя столбца> [, <имя столбца> ...]

<значения> ::= { <значение> | DEFAULT} [, { <значение> | DEFAULT} ...]

<список возвращаемых выражений> ::= <выражение> [[AS] <псевдоним>] [, <выражение>
[[AS] <псевдоним>] ...]

<список переменных> ::= [:] <имя переменной> [, [:] <имя переменной> ...]
```

Оператор MERGE производит слияние записей источника в целевую таблицу (или обновляемое представление). Источником данных может быть таблица, представление, хранимая процедура или производная таблица, т.е. заключенный в скобки оператор SELECT. Каждая запись источника используется для обновления (предложение UPDATE) или удаления (предложение DELETE) одной или более записей цели, или вставки (предложение INSERT) записи в целевую таблицу, или ни для того, ни для другого. Условие обычно содержит сравнение столбцов в таблицах источника и цели.

Подробнее об операторе слияния см. в главе 11.

RECREATE EXCEPTION

Оператор создает новое пользовательское исключение, если оно отсутствует в базе данных, или пересоздает существующее. Синтаксис оператора:

Листинг Ж.87. Синтаксис оператора RECREATE EXCEPTION

```
RECREATE EXCEPTION <имя исключения>  '<текст сообщения>';
```

Имя исключения может содержать до 63 символов и должно быть уникальным среди всех имен исключений базы данных.

Текст сообщения — текст, выдаваемый в момент вызова исключения. Может содержать до 1021 символа.

Оператор нормально выполняется только если это исключение не используется в каком-либо триггере или хранимой процедуре.

См. также операторы ALTER EXCEPTION, DROP EXCEPTION, CREATE EXCEPTION, CREATE OR ALTER EXCEPTION, операторы PSQL WHEN-DO, EXCEPTION.

RECREATE FUNCTION

Оператор `RECREATE FUNCTION` позволяет создать новую хранимую функцию, если функция с тем же именем отсутствует в базе данных, или пересоздать существующую в базе данных функцию. Если функция с этим именем уже существует, то оператор попытается удалить её и создать новую функцию, при этом существующие привилегии и зависимости не сохраняются. Синтаксис оператора:

Листинг Ж.88. Синтаксис оператора создания новой или изменения существующей хранимой функции RECREATE FUNCTION

```
RECREATE FUNCTION <имя хранимой функции>
   [(<входной параметр> [, <входной параметр> ...])]
RETURNS <тип> [COLLATE <сортировка>] [DETERMINISTIC]
[SQL SECURITY {DEFINER | INVOKER}]
{ EXTERNAL NAME '<внешний модуль>' ENGINE <имя движка> [AS <тело внешней функции>]}
|
{
   {AS '<BLR код>'}
   |
   {AS
      [<объявление> [<объявление> ...] ]
   BEGIN
      <блок операторов>
   END }
}
```

Операция закончится неудачей при подтверждении транзакции, если функция имеет зависимости.

См. также операторы CREATE OR ALTER FUNCTION, CREATE FUNCTION, ALTER FUNCTION, DROP FUNCTION, DECLARE VARIABLE, DECLARE CURSOR, DECLARE FUNCTION, DECLARE PROCEDURE.

RECREATE PACKAGE

Оператор `RECREATE PACKAGE` создаёт новый или пересоздает существующий заголовок пакета. Синтаксис оператора:

Листинг Ж.89. Синтаксис оператора RECREATE PACKAGE

```
RECREATE PACKAGE <имя пакета>
[SQL SECURITY {DEFINER | INVOKER}]
AS
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
BEGIN
   [ <объявление процедуры> | <объявление функции> ...]
END

<объявление процедуры> ::=
   PROCEDURE <имя процедуры> [(<входной параметр> [, <входной параметр> ...])]
      [RETURNS (<выходной параметр> [, <выходной параметр> ...])]

<объявление функции> ::=
   FUNCTION <имя функции> [(<входной параметр> [, <входной параметр> ...])]
      RETURNS <тип> [COLLATE <сортировка>] [DETERMINISTIC]
```

Если заголовок пакета с таким именем уже существует, то оператор попытается удалить его и создать новый заголовок пакета. Пересоздать заголовок пакета невозможно, если у существующей заголовка пакета имеются зависимости или существует тело этого пакета. После пересоздания заголовка пакета привилегии на выполнение подпрограмм пакета и привилегии самого пакета не сохраняются.

См. также операторы RECREATE PACKAGE BODY, CREATE PACKAGE, DROP PACKAGE, ALTER PACKAGE, CREATE OR ALTER PACKAGE, CREATE PROCEDURE, CREATE FUNCTION.

RECREATE PACKAGE BODY

Оператор `RECREATE PACKAGE BODY` создаёт новое или пересоздает существующее тело пакета. Синтаксис оператора:

Листинг Ж.90. Синтаксис оператора `RECREATE PACKAGE BODY`

```
RECREATE PACKAGE BODY <имя пакета>
AS
BEGIN
   [ <объявление процедуры> | <объявление функции> ...]
   [ <реализация процедуры> | <реализация функции> ...]
END

<объявление процедуры> ::=
   PROCEDURE <имя процедуры> [(<входной параметр> [, <входной параметр> ...])]
      [RETURNS (<выходной параметр> [, <выходной параметр> ...])]

<объявление функции> ::=
   FUNCTION <имя функции> [(<входной параметр> [, <входной параметр> ...])]
      RETURNS <тип> [COLLATE <сортировка>] [DETERMINISTIC]

<реализация процедуры> ::=
   PROCEDURE <имя процедуры> [(<входной_параметр> [, <входной_параметр> ...])]
      [RETURNS (<выходной параметр> [, <выходной параметр> ...])]
      {EXTERNAL NAME '<внешний модуль>' ENGINE <имя движка> [AS <тело вн. проц.>]} |
      { AS
         [<объявление> [<объявление> ...] ]
      BEGIN
         <блок операторов>
      END }
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
<реализация функции> ::=
   FUNCTION <имя функции> [(<входной_параметр> [, <входной_параметр> ...])]
      RETURNS <тип> [COLLATE <сортировка>] [DETERMINISTIC]
      {EXTERNAL NAME '<внешний модуль>' ENGINE <имя движка> [AS <тело вн. функ.>]} |
      { AS
         [<объявление> [<объявление> ...] ]
      BEGIN
         <блок операторов>
      END }
```

Если тело пакета с таким именем уже существует, то оператор попытается удалить его и создать новое тело пакета. После пересоздания тела пакета привилегии на выполнение подпрограмм пакета и привилегии самого пакета сохраняются.

См. также операторы RECREATE PACKAGE, CREATE PACKAGE BODY, DROP PACKAGE BODY, CREATE PROCEDURE, CREATE FUNCTION.

RECREATE PROCEDURE

Оператор `RECREATE PROCEDURE` создает новую или пересоздает существующую хранимую процедуру. Если процедура с таким именем уже существует, то оператор попытается удалить ее и создать новую процедуру, при этом привилегии на выполнение хранимой процедуры и привилегии самой хранимой процедуры не сохраняются. Синтаксис оператора:

Листинг Ж.91. Синтаксис оператора `RECREATE PROCEDURE`

```
RECREATE PROCEDURE <имя хранимой процедуры>
[AUTHID {OWNER | CALLER}]
   [(<входной параметр> [, <входной параметр> ...])]
[RETURNS (<выходной параметр> [, <выходной параметр> ...])]
[SQL SECURITY {DEFINER | INVOKER}]
{ EXTERNAL NAME '<внешний модуль>' ENGINE <имя движка> [AS <тело внешней проц.>]} |
{
   {AS '<BLR код>'}
   |
   {AS
      [<объявление> [<объявление> ...] ]
   BEGIN
      <блок операторов>
   END }
}

<входной параметр> ::= <описание параметра> [{=|DEFAULT} <значение по умолчанию>]

<выходной параметр> ::= <описание параметра>

<описание параметра> ::= <имя параметра> <тип> [NOT NULL] [COLLATE <порядок
сортировки>]

<тип> ::= { <тип данных SQL>
          | [TYPE OF] <имя домена>
          | TYPE OF COLUMN <имя таблицы/представления>.<имя столбца> }
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
<значение по умолчанию>::= {<литерал> | NULL | <контекстная переменная>}

<внешний модуль> ::= '<имя внешнего модуля>!<имя функции в модуле>[! <информация>]'

<объявление> ::= <объявление локальной переменной>;
                 | <объявление курсора>;
                 | <объявление процедуры/функции>
                 | <реализация процедуры/функции>
```

Синтаксис и семантика оператора (за исключением названия оператора) полностью соответствует оператору `CREATE PROCEDURE`.

Операция закончится неудачей при подтверждении транзакции, если процедура имеет зависимости.

См. также операторы CREATE OR ALTER PROCEDURE, CREATE PROCEDURE, ALTER PROCEDURE, DROP PROCEDURE, EXECUTE PROCEDURE, DECLARE VARIABLE, DECLARE CURSOR, DECLARE FUNCTION, DECLARE PROCEDURE.

RECREATE SEQUENCE

Последовательность можно пересоздать с помощью оператора `RECREATE GENERATOR (SEQUENCE)`:

Листинг Ж.92. Синтаксис оператора `RECREATE GENERATOR /SEQUENCE`

```
RECREATE {GENERATOR | SEQUENCE} <имя генератора>
  [START WITH <начальное значение>] [INCREMENT [BY] <приращение>];
```

Если последовательность с таким именем уже существует, то оператор `RECREATE SEQUENCE` попытается удалить её и создать новую последовательность. При наличии зависимостей для существующей последовательности оператор `RECREATE SEQUENCE` не выполнится.

RECREATE TABLE

Таблица пересоздается оператором `RECREATE TABLE`.

Листинг Ж.93. Синтаксис оператора создания таблицы `RECREATE TABLE`

```
RECREATE [GLOBAL TEMPORARY] TABLE <имя таблицы>
   [EXTERNAL [FILE] '<спецификация файла>']
   (<определение столбца> [, { <определение столбца> | <ограничение таблицы>}...])
   [ON COMMIT {DELETE | PRESERVE} ROWS]
   [SQL SECURITY {DEFINER | INVOKER}];
```

Этот оператор создаёт или пересоздает таблицу. Если таблица с таким именем уже существует, то оператор `RECREATE TABLE` попытается удалить её и создать новую. Оператор `RECREATE TABLE` не выполнится, если существующая таблица имеет зависимости.

Данная операция доступна и для глобальных временных таблиц (GTTs) с синтаксисом, аналогичным оператору `CREATE GLOBAL TEMPORARY TABLE`.

См. также операторы CREATE TABLE, ALTER TABLE, DROP TABLE.

RECREATE TRIGGER

Оператор RECREATE TRIGGER создаёт новый триггер, если триггер с указанным именем не существует, в противном случае оператор RECREATE TRIGGER попытается удалить его и создать новый. Синтаксис оператора:

Листинг Ж.94. Синтаксис оператора RECREATE TRIGGER

```
RECREATE TRIGGER <имя триггера> {
   <объявление табличного триггера>
 | <объявление табличного триггера в стандарте SQL-2003>
 | <объявление триггера базы данных>
 | <объявление DDL триггера> }
[SQL SECURITY {DEFINER | INVOKER}]
{ EXTERNAL NAME '<внешний модуль>' ENGINE <имя движка> [AS <тело внеш. триг.>] } |
{
   AS
      [<объявление> [<объявление> ...] ]
   BEGIN
      <блок операторов>
   END }

   <объявление табличного триггера> ::=
         FOR {<имя таблицы> | <имя представления>}
         [ACTIVE | INACTIVE]
         {BEFORE | AFTER} <список событий таблицы (представления)>
         [POSITION <порядок срабатывания триггера>]

<объявление табличного триггера в стандарте SQL-2003> ::=
         [ACTIVE | INACTIVE]
         {BEFORE | AFTER} <список событий таблицы (представления)>
         [POSITION <порядок срабатывания триггера>]
         ON {<имя таблицы> | <имя представления>}

<объявление триггера базы данных> ::=
         [ACTIVE | INACTIVE]
         ON <событие соединения или транзакции>
         [POSITION <порядок срабатывания триггера>]

<объявление DDL триггера> ::=
         [ACTIVE | INACTIVE]
         {BEFORE | AFTER} <список DDL событий>
         [POSITION <порядок срабатывания триггера>]

<список событий таблицы (представления)> ::= <событие DML> [OR <событие DML>...]

<событие DML> ::= { INSERT | UPDATE | DELETE }

<событие соединения или транзакции> ::= {
           CONNECT
         | DISCONNECT
         | TRANSACTION START
         | TRANSACTION COMMIT
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
          | TRANSACTION ROLLBACK }

<список DDL событий> ::= {
            ANY DDL STATEMENT
          | <DDL событие> [OR <DDL событие> ...] }

<DDL событие> ::=
            CREATE|ALTER|DROP TABLE
          | CREATE|ALTER|DROP PROCEDURE
          | CREATE|ALTER|DROP FUNCTION
          | CREATE|ALTER|DROP TRIGGER
          | CREATE|ALTER|DROP EXCEPTION
          | CREATE|ALTER|DROP VIEW
          | CREATE|ALTER|DROP DOMAIN
          | CREATE|ALTER|DROP ROLE
          | CREATE|ALTER|DROP SEQUENCE
          | CREATE|ALTER|DROP USER
          | CREATE|ALTER|DROP INDEX
          | CREATE|DROP COLLATION
          | ALTER CHARACTER SET
          | CREATE|ALTER|DROP PACKAGE
          | CREATE|DROP PACKAGE BODY
          | CREATE|ALTER|DROP MAPPING

<объявление> ::= <объявление локальной переменной>;
                 | <объявление курсора>;
                 | <объявление процедуры/функции>
                 | <реализация процедуры/функции>
```

Синтаксис и семантика оператора (за исключением названия оператора) полностью соответствует оператору `CREATE TRIGGER`.

См. также операторы CREATE OR ALTER TRIGGER, CREATE TRIGGER, ALTER TRIGGER, DROP TRIGGER, DECLARE VARIABLE.

RECREATE USER

Оператор `RECREATE USER` создаёт нового или пересоздаёт существующего пользователя.

Листинг Ж.95. Синтаксис оператора `RECREATE USER`

```
RECREATE USER <логин> PASSWORD '<пароль>'
   [FIRSTNAME '<имя пользователя>']
   [MIDDLENAME '<отчество пользователя>']
   [LASTNAME '<фамилия пользователя>']
   [ACTIVE | INACTIVE]
   [TAGS (<атрибут>|DROP <имя атрибута> [, <атрибут>|DROP <имя атрибута>...] )]
   [USING PLUGIN 'имя плагина']
   [GRANT ADMIN ROLE];

<атрибут> ::= <имя атрибута> = 'строковое значение'
```

Если пользователь с таким именем уже существует, то оператор `RECREATE USER` удалит его и

создаст нового. Существующие привилегии при этом будут сохранены.

> Если предложение `USING PLUGIN` не указано, то при добавлении пользователя он сам добавляется во все плагины из списка параметра `DefaultUserManagers` (в том числе атрибуты).

Семантика операторов и предложений в этом операторе полностью соответствует оператору `CREATE USER`.

См. также операторы CREATE USER, DROP USER.

RECREATE VIEW

Оператор `RECREATE VIEW` позволяет внести изменения в существующее представление. Его синтаксис:

Листинг Ж.96. Синтаксис оператора `RECREATE VIEW`

```
RECREATE VIEW <имя представления> [<список столбцов>]
AS <оператор SELECT>
[WITH CHECK OPTION]

<список столбцов> ::= (столбец [, столбец ...])
```

Представление может отсутствовать в базе данных. В этом случае оно просто создается заново. Если представление с этим именем уже существует в базе данных, то оно удаляется и затем создается заново. Попытка выполнить пересоздание представления приведет к ошибке базы данных, если это представление в настоящий момент находится в использовании.

Все предложения в этом операторе в точности соответствуют предложениям в операторе `CREATE VIEW`.

См. также операторы CREATE VIEW, DROP VIEW, SELECT, INSERT, UPDATE, UPDATE OR INSERT, DELETE.

RELEASE SAVEPOINT

Оператор удаляет созданную ранее точку сохранения. Синтаксис:

Листинг Ж.97. Синтаксис оператора `RELEASE SAVEPOINT`

```
RELEASE SAVEPOINT <имя точки сохранения> [ONLY];
```

Точка сохранения с указанным именем удаляется из списка точек сохранения, созданных транзакцией. Если не задано ключевое слово `ONLY`, то удаляются и все последующие точки сохранения. Если указано ключевое слово `ONLY`, то удаляется из списка только заданная точка сохранения.

Если точки сохранения с указанным именем в списке не существует, то никакие действия не выполняются.

Подробно оператор описан в главе 5.

См. также операторы SET TRANSACTION, ROLLBACK, SAVEPOINT, COMMIT.

REVOKE

Оператор позволяет отменить привилегии для пользователей, ролей, хранимых процедур, хранимых функций, пакетов, триггеров и представлений, которые были предоставлены оператором `GRANT`. Синтаксис оператора `REVOKE`:

Листинг Ж.98. Синтаксис оператора `REVOKE`

```
REVOKE [GRANT OPTION FOR] <табличные привилегии>
   ON [TABLE] {<имя таблицы> | <имя представления>}
   FROM <список обладателей привилегий>
   [{GRANTED BY|AS} [USER] <имя грантора>]

REVOKE [GRANT OPTION FOR] EXECUTE ON {PROCEDURE | FUNCTION | PACKAGE} <имя процедуры/
функции/пакета>
   FROM <список обладателей привилегий>
   [{GRANTED BY|AS} [USER] <имя грантора>]

REVOKE [GRANT OPTION FOR] USAGE ON {EXCEPTION <имя искл-я>| {GENERATOR | SEQUENCE}
<имя генератора>}
   FROM <список обладателей привилегий>
   [{GRANTED BY|AS} [USER] <имя грантора>]

REVOKE [GRANT OPTION FOR] {ALL [PRIVILEGES] | {CREATE|ALTER ANY|DROP ANY} [,
{CREATE|ALTER ANY|DROP ANY}...] } <объект>
   FROM <список обладателей привилегий>
   [{GRANTED BY|AS} [USER] <имя грантора>]

REVOKE CREATE DATABASE FROM <пользователь обладатель> {,<пользователь обладатель>}

REVOKE [GRANT OPTION FOR] {ALL [PRIVILEGES]|{ALTER|DROP}[,{ALTER|DROP}]} DATABASE
   FROM <список обладателей привилегий>
   [{GRANTED BY|AS} [USER] <имя грантора>]

REVOKE [ADMIN OPTION FOR] [DEFAULT] <имя роли> [, [DEFAULT] <имя роли> ...]
   FROM [USER]|[ROLE] <имя польз-я/роли> [, [USER]|[ROLE] <имя польз-я/роли>...]
   [{GRANTED BY|AS} [USER] <имя грантора>]

REVOKE ALL ON ALL FROM <список обладателей привилегий>

<табличные привилегии> ::= ALL [PRIVILEGES] | <привилегия> [, <привилегия>...]

<привилегия> ::=  SELECT [(<имя столбца [, <имя столбца>...]>)]
                | DELETE
                | INSERT
                | UPDATE [(<имя столбца [, <имя столбца>...]>)]
                | REFERENCES [(<имя столбца [, <имя столбца>...]>)] }

<объект> ::=  TABLE
            | TABLESPACE
            | VIEW
            | PROCEDURE
            | FUNCTION
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
              | PACKAGE
              | GENERATOR
              | SEQUENCE
              | DOMAIN
              | EXCEPTION
              | ROLE
              | CHARACTER SET
              | COLLATION
              | FILTER
              | JOB

<список обладателей привилегий> ::={<объект обладатель>|<пользователь обладатель>}
                                [, {<объект обладатель>|<пользователь обладатель>}...]

<объект обладатель> ::=  PROCEDURE <имя процедуры>
                       | FUNCTION <имя функции>
                       | PACKAGE <имя пакета>
                       | TRIGGER <имя триггера>
                       | VIEW <имя представления>
                       | SYSTEM PRIVILEGE <системная привилегия>

<пользователь обладатель> ::=  [USER] <имя пользователя>
                             | [ROLE] <имя роли>
                             | GROUP <имя группы в Unix>
```

Только пользователь, который назначил привилегию, может удалить её.

Оператор позволяет выполнить одно из следующих действий:

- отнять одну или несколько привилегий для таблиц и представлений у пользователей, ролей, хранимых процедур, хранимых функций, пакетов, триггеров и представлений;
- отнять права на выполнение процедуры, функции или пакета у пользователей, ролей, хранимых процедур, хранимых функций, пакетов, триггеров и представлений;
- отнять привилегии на контроль доступа к исключениям, последовательностям и генераторам;
- отнять одну или несколько привилегий на выполнение DDL операций над основными объектами базы данных у пользователей, ролей, хранимых процедур, хранимых функций, пакетов, триггеров и представлений;
- отнять привилегии на создание, удаление и изменение базы данных у пользователей, ролей, хранимых процедур, хранимых функций, пакетов, триггеров и представлений;
- отменить назначенные оператором `GRANT` роли;
- отменить все привелегии на всех объектах у пользователей.

Подробное описание оператора см. в главе 26.

См. также операторы CREATE ROLE, DROP ROLE, GRANT, CONNECT.

ROLLBACK

Оператор позволяет выполнить отмену всех действий, выполненных в контексте данной транзакции, или откат на ранее созданную в транзакции контрольную точку, точку сохранения (в случае использования вложенных транзакций), если задано необязательное предложение `TO SAVEPOINT`. Синтаксис оператора:

Листинг Ж.99. Синтаксис оператора ROLLBACK

```
ROLLBACK  [WORK] [TRANSACTION <имя транзакции>]
[RETAIN [SNAPSHOT] | TO SAVEPOINT <имя точки сохранения>] [RELEASE];
```

Подробно оператор, работа с транзакциями, использование вложенных транзакций описаны в главе 5.

См. также операторы SET TRANSACTION, COMMIT, SAVEPOINT, RELEASE SAVEPOINT.

SAVEPOINT

Оператор создает очередную точку сохранения для транзакции, на которую в дальнейшем возможен возврат в процессе выполнения действий в контексте данной транзакции. Синтаксис оператора:

Листинг Ж.100. Синтаксис оператора создания точки сохранения SAVEPOINT

```
SAVEPOINT <имя точки сохранения>;
```

Имя точки сохранения — идентификатор базы данных, который может содержать до 63 символов. Имена точек сохранения в контексте одной транзакции должны отличаться. Если же в операторе создается точка сохранения с именем, уже присутствующем в списке созданных точек сохранения в процессе активности данной транзакции, то новое состояние базы данных заменяет старую точку сохранения, все последующие точки удаляются.

Подробно оператор, работа с транзакциями, использование вложенных транзакций описаны в главе 5.

См. также операторы SET TRANSACTION, ROLLBACK, COMMIT, RELEASE SAVEPOINT.

SELECT

Оператор SELECT дает возможность осуществлять довольно сложную выборку данных из одной или более таблиц базы данных. Он позволяет выполнять объединение (UNION) и соединение (JOIN) данных из различных таблиц. Синтаксис оператора SELECT:

Листинг Ж.101. Синтаксис оператора SELECT

```
[WITH [RECURSIVE] <CTE> [, <CTE> ...]]
SELECT
   [FIRST <значение>] [SKIP <значение>]
   [DISTINCT | ALL]
   <выходное поле> [, <выходное поле>]
FROM
   <источники>
   [<соединения (joins)>]
[WHERE <условие выборки>]
[GROUP BY <условие группирование выбранных данных>
   [HAVING <условие выборки>]]
[WINDOW <спецификация окна> [, <спецификация окна>] ...]
[UNION [DISTINCT | ALL] <другой набор данных>]
[PLAN <выражение для плана поиска>]
[ORDER BY <выражение для порядка выборки>]
[OPTIMIZE FOR {FIRST | ALL} ROWS]
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
[   ROWS <m> [TO <n>]
 | [OFFSET <n> {ROW | ROWS}] [FETCH {FIRST | NEXT} [<m>] {ROW | ROWS} ONLY] ]
[FOR UPDATE [OF <имя столбца> [, <имя столбца>]...]]
[WITH LOCK [SKIP LOCKED]]
[INTO [:]<переменная> [,[:]<переменная> ... ]]
```

- Предложение `WITH` позволяет задать общее табличное выражение (CTE, Common Table Expression). Оно может быть рекурсивным (ключевое слово `RECURSIVE`) и обычным, не рекурсивным (значение по умолчанию).
- Предложения `FIRST` и `SKIP` ограничивают результирующий набор данных указанным числом записей.
- Ключевое слово `DISTINCT` указывает, что в выходной набор данных не помещаются дубликаты строк.
- Предложение `FROM` содержит список таблиц, из которых осуществляется выбор данных.
- Необязательное предложение `WHERE` задает условия выборки данных — условия, которым должны удовлетворять строки исходной таблицы (исходных таблиц), для того, чтобы они попали в результирующий набор данных.
- Предложения `GROUP BY` и `HAVING` позволяют сгруппировать выбранные данные, если в списке выбора присутствуют агрегатные функции.
- Предложение `UNION` дает возможность объединить в выходном наборе данных несколько таблиц с одинаковой структурой.
- В предложении `PLAN` можно задать свой план для выполнения запроса, который позволит ускорить процесс выбора данных.
- Предложение `ORDER BY` задает упорядоченность выходного набора данных. Здесь также можно указать количество строк, которое должно быть помещено в результирующий набор данных (предложения `ROWS, OFFSET, FETCH`).
- Предложение `OPTIMIZE FOR` позволяет задать необходимую стратегию оптимизации запросов для ускорения процесса выборки данных.
- Предложение `WITH LOCK` запрещает параллельным процессам, транзакциям выполнять какие-либо изменения в данной таблице запроса. Подробности см. в главе 5.
- Предложение `SKIP LOCKED` позволяет пропускать записи, заблокированные другими транзакциями, вместо того чтобы ждать их разблокирования или выдавать ошибку конфликта обновления.
- С помощью предложения `INTO` в PSQL (хранимых процедурах, триггерах и др.) результаты выборки команды `SELECT` могут быть построчно загружены в локальные переменные (число, порядок и типы локальных переменных должны соответствовать полям `SELECT`).

Подробно об операторе `SELECT` и примеры использования см. в разделе 11.1.

См. также операторы INSERT, DELETE, UPDATE, UPDATE OR INSERT, SET TRANSACTION, EXECUTE PROCEDURE.

Предложение OPTIMIZE FOR

Предложение `OPTIMIZE FOR` позволяет задать необходимую стратегию оптимизации запросов, тем самым ускорить процесс выборки данных. Синтаксис предложения:

```
[OPTIMIZE FOR {FIRST | ALL} ROWS]
```

- `FIRST` - для запросов выбирается такой план доступа, который позволяет максимально быстро получить первые записи в выборке;
- `ALL` - для запросов выбирается такой план доступа, который позволяет максимально быстро

получить все записи в выборке.

В отсутсвии предложения OPTIMIZE FOR при выполнении оператора SELECT будет использоваться стратегия оптимизации запросов, указанная в параметре конфигурационного файла OptimizationStrategy. Значением по умолчанию является default. Если при умолчательном значении параметра OptimizationStrategy в запросе присутствуют ключевые слова FIRST и/или SKIP или же предложение ROWS, то будет использована стратегия оптимизации FIRST ROWS, несмотря на настройки файла конфигурации. Если же в конфигурационном файле указана стратегия ALL ROWS, то данные предложения не будут влиять на оптимизацию запросов. При стратегии ALL ROWS всегда применяется явный выбор данных и их сортировка.

SET BIND

Оператор SET BIND настраивает правила приведения к типу данных для текущего сеанса. Этот оператор позволяет заменять один тип данных другим при выполнении клиент-серверных взаимодействий.

Листинг Ж.102. Синтаксис оператора SET BIND

```
SET BIND OF <заменяемый тип> TO <итоговый тип>

<заменяемый тип> ::= <scalar_datatype>
                   | <blob_datatype>
                   | TIME ZONE
                   | VARCHAR | {CHARACTER | CHAR} VARYING

<итоговый тип> ::= <scalar_datatype>
                 | <blob_datatype>
                 | VARCHAR | {CHARACTER | CHAR} VARYING
                 | LEGACY | NATIVE | EXTENDED
                 | EXTENDED TIME WITH TIME ZONE
                 | EXTENDED TIMESTAMP WITH TIME ZONE
```

Если используется неполное определение типа (например, CHAR вместо CHAR(n)) в типе данных, который нужно заменить, то принудительное преобразование выполняется для всех столбцов CHAR. Специальный неполный тип TIME ZONE обозначает TIME WITH TIME ZONE и TIMESTAMP WITH TIME ZONE. Если в типе данных, к которому нужно преобразовать, используется неполное определение типа, сервер автоматически определяет недостающие сведения об этом типе на основе исходного столбца.

Изменение привязки типа данных NUMERIC или DECIMAL не влияет на базовый целочисленный тип. Напротив, изменение привязки целочисленного типа данных также влияет на соответствующие NUMERIC или DECIMAL (например, SET BIND OF INT128 TO DOUBLE PRECISION также будет преобразовывать NUMERIC и DECIMAL с точностью более 19, поскольку они используют INT128 в качестве базового типа).

Специальный тип LEGACY используется, когда тип данных, отсутствующий в предыдущей версии СУБД, должен быть представлен способом, понятным для устаревшего клиентского программного обеспечения (возможно, с некоторой потерей данных). Правила приведения, применяемые в этом случае, описаны в таблице ниже.

Таблица Ж.1 — Правила преобразования NATIVE к LEGACY

NATIVE	LEGACY
BOOLEAN	CHAR(5)

(разрыв таблицы)

(разрыв таблицы)

NATIVE	LEGACY
DECFLOAT	DOUBLE PRECISION
INT128	BIGINT
TIME WITH TIME ZONE	TIME WITHOUT TIME ZONE
TIMESTAMP WITH TIME ZONE	TIMESTAMP WITHOUT TIME ZONE

Установка преобразования к NATIVE сбрасывает существующее правило преобразования для этого типа данных и он будет передаваться в исходном формате.

Выполнение ALTER SESSION RESET приведет к возвращению к правилам преобразования по умолчанию.

Пример работы оператора SET BIND:

```
-- native
SELECT CAST('123.45' AS DECFLOAT(16)) FROM RDB$DATABASE;

CAST
======================
123.45

-- double
SET BIND OF DECFLOAT TO DOUBLE PRECISION;
SELECT CAST('123.45' AS DECFLOAT(16)) FROM RDB$DATABASE;

CAST
======================
123.4500000000000

-- still double
SET BIND OF DECFLOAT(34) TO CHAR;
SELECT CAST('123.45' AS DECFLOAT(16)) FROM RDB$DATABASE;

CAST
======================
123.4500000000000

-- text
SELECT CAST('123.45' AS DECFLOAT(34)) FROM RDB$DATABASE;

CAST
======================
123.45
```

SET DECFLOAT BIND

Для того чтобы старые приложения умели работать с типом DECFLOAT можно настроить отображение значений DECFLOAT на другие доступные типы данных с помощью оператора SET DECFLOAT BIND.

Листинг Ж.103. Синтаксис оператора SET DECFLOAT BIND

```
SET DECFLOAT BIND <тип для привязки>
<тип для привязки> ::= NATIVE
                     | CHAR[ACTER]
                     | DOUBLE PRECISION
                     | BIGINT[, <точность>]
```

Допустимые типы для привязки:

- `NATIVE` — используется IEEE754 двоичное представление. Идеальная поддержка для дальнейшей обработки, но с плохой точностью
- `CHAR[ACTER]` — ASCII строка. Идеальная точность, но плохая поддержка для дальнейшей обработки.
- `DOUBLE PRECISION` — используется 8 байтное представление с плавающей точкой, тоже самое что и для полей типа `DOUBLE PRECISION`. Идеальная поддержка для дальнейшей обработки, но с плохой точностью.
- `BIGINT` — используется целое с указанным масштабом, тоже самое что поле `NUMERIC(18, <точность>)`. Хорошая поддержка дальнейшей обработки и требуемая точность, но диапазон значений очень ограничен.

Привязка к ASCII строке будет иметь тип `CHAR(23)` для `DECFLOAT(16)` и `CHAR(42)` для `DECFLOAT(34)`.

Строковое представление зависит от значения `DECFLOAT`: если оно является экспоненциальным, а требования к точности позволяют отображать значение без использования научной записи, используется полностью выписанный формат.

Данный SQL оператор работает вне механизма управления транзакциями, изменения выполненные им вступаю в силу немедленно. Его использование разрешено в том числе и PSQL коде.

SET DECFLOAT ROUND

Оператор позволяет изменять режим округления для типа `DECFLOAT` для текущей сессии.

Листинг Ж.104. Синтаксис оператора SET DECFLOAT ROUND

```
SET DECFLOAT ROUND <режим округления>
```

Поддерживаются следующие режимы округления совместимые со стандартом IEEE:

Таблица Ж.2 — Режим округления

Режим округления	Описание
CEILING	Округление сверху. Если все отбрасываемые цифры равны нулю или знак числа отрицателен, последняя неотбрасываемая цифра не меняется. В противном случае последняя неотбрасываемая цифра инкрементируется на единицу (округляется в большую сторону).
UP	Округление по направлению от нуля (усечение с приращением). Отбрасываемые значения игнорируются.
HALF_UP	Округление к ближайшему значению. Если результат равноудаленный, выполняется округление в большую сторону. Если отбрасываемые значения больше чем или равны половине (0,5) единицы в следующей левой позиции, последняя неотбрасываемая цифра инкрементируется на единицу (округляется в большую сторону). В противном случае отбрасываемые значения игнорируются. Этот режим выбран по умолчанию.
HALF_EVEN	Округление к ближайшему значению. Если результат равноудаленный, выполняется округление так, чтобы последняя цифра была четной. Если отбрасываемые значения больше половины (0,5) единицы в следующей левой позиции, последняя неотбрасываемая цифра инкрементируется на единицу (округляется в большую сторону). Если они меньше половины, результат не корректируется (то есть отбрасываемые знаки игнорируются). В противном случае, когда отбрасываемые значения точно равны половине, последняя неотбрасываемая цифра не меняется, если она является четной и инкрементируется на единицу (округляется в большую сторону) в противном случае (чтобы получить четную цифру). Этот режим округления называется также банковским округлением и дает ощущение справедливого округления.
HALF_DOWN	Округление к ближайшему значению. Если результат равноудаленный, выполняется округление в меньшую сторону. Если отбрасываемые значения больше чем или равны половине (0,5) единицы в следующей левой позиции, последняя неотбрасываемая цифра декрементируется на единицу (округляется в меньшую сторону). В противном случае отбрасываемые значения игнорируются.
DOWN	Округление по направлению к нулю (усечение). Отбрасываемые значения игнорируются.
FLOOR	Округление снизу. Если все отбрасываемые цифры равны нулю или знак положителен, последняя неотбрасываемая цифра не меняется. В противном случае (знак отрицателен) последняя неотбрасываемая цифра инкрементируется на единицу.
REROUND	Округление к большему значению, если округляется 0 или 5, в противном случае округление происходит к меньшему значению.

SET DECFLOAT TRAPS TO

В процессе вычисления выражений могут возникнуть различные ситуации, которые могут вызвать исключение или проигнорированы. Установить какие ситуации приведут к возбуждению исключения можно с помощью оператора `SET DECFLOAT TRAPS TO`.

Листинг Ж.105. Синтаксис оператора `SET DECFLOAT TRAPS TO`

```
SET DECFLOAT TRAPS TO <ситуация>[, <ситуация>...]
<ситуация> ::= Division_by_zero
             | Inexact
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
                | Invalid_operation
                | Overflow
                | Underflow
```

По умолчанию исключения генерируется для следующих ситуаций: `Division_by_zero, Invalid_operation, Overflow`.

Данный SQL оператор работает вне механизма управления транзакциями, изменения выполненные им вступаю в силу немедленно. Его использование разрешено в том числе и PSQL коде.

SET GENERATOR

Оператор позволяет задать новое значение для генератора. Синтаксис:

Листинг Ж.106. Синтаксис оператора SET GENERATOR

```
SET GENERATOR <имя генератора> TO <значение>;
```

Существует оператор `ALTER SEQUENCE`, который позволяет выполнить те же действия. Использование оператора `ALTER SEQUENCE` является более предпочтительным.

Оператор `SET GENERATOR` может выполнить владелец последовательности, администратор и пользователь с привилегией `ALTER ANY SEQUENCE (ALTER ANY GENERATOR)`.

Подробно оператор описан в главе 12.

См. также операторы CREATE GENERATOR, CREATE SEQUENCE, DROP GENERATOR, DROP SEQUENCE, ALTER SEQUENCE, функцию GEN_ID, конструкцию NEXT VALUE FOR.

SET NAMES

Оператор устанавливает набор символов для клиентской стороны соединения с базой данных. Синтаксис оператора:

Листинг Ж.107. Синтаксис оператора SET NAMES

```
SET NAMES <набор символов>;
```

Оператор должен быть выполнен до выполнения оператора соединения с базой данных `CONNECT`. Если оператор не выдавался, то для клиента будет установлен набор символов `NONE`, что при дальнейшей работе с базой данных (с символьными данными) может создать массу проблем. Использование оператора позволяет серверу базы данных выполнять трансляцию (преобразование) символьных данных между набором символов базы данных и набором символов клиента.

Хорошей практикой является использование на клиенте того же набора символов, что и набор символов по умолчанию для базы данных `DEFAULT CHARACTER SET`.

Список наборов символов представлен в Приложение Д.

См. также операторы CONNECT, CREATE DATABASE, SET SQL DIALECT.

SET ROLE

Согласно стандарту SQL-2008 оператор `SET ROLE` позволяет установить контекстной переменной `CURRENT_ROLE` одну из назначенных ролей для пользователя `CURRENT_USER` или роль, полученную в результате доверительной аутентификации (в этом случае оператор принимает вид `SET TRUSTED ROLE`). Синтаксис оператора:

Листинг Ж.108. Синтаксис оператора `SET ROLE`

```
SET ROLE <имя роли>;
```

См. также оператор SET TRUSTED ROLE.

SET SESSION IDLE TIMEOUT

Оператор позволяет устанавливать тайм-аут простоя соединения на уровне текущего соединения.

Листинг Ж.109. Синтаксис оператора `SET SESSION IDLE TIMEOUT`

```
SET SESSION IDLE TIMEOUT <значение> [HOUR | MINUTE | SECOND]
```

В качестве параметра выступает значение тайм-аута простоя в указанных единицах измерения времени. Если единица измерения времени не указано, то по умолчанию значение тайм-аута измеряется в минутах.

Данный SQL оператор работает вне механизма управления транзакциями и вступают в силу немедленно.

```
SET SESSION IDLE TIMEOUT 8 HOUR;
```

SET SQL DIALECT

Оператор задает диалект клиента для выполнения доступа к базе данных.

Листинг Ж.110. Синтаксис оператора `SET SQL DIALECT`

```
SET SQL DIALECT {1 | 3};
```

Оператор должен быть выполнен до выполнения оператора соединения с базой данных `CONNECT`. Значением диалекта должен быть именно тот диалект, который был использован при создании базы данных, иначе будет выдано диагностическое сообщение.

См. также операторы CONNECT, CREATE DATABASE, SET NAMES.

SET STATEMENT TIMEOUT

Оператор позволяет установить тайм-аут выполнения SQL операторов на уровне текущего соединения.

Листинг Ж.111. Синтаксис оператора SET STATEMENT TIMEOUT

```
SET STATEMENT TIMEOUT <значение> [HOUR | MINUTE | SECOND | MILLISECOND]
```

В качестве параметра выступает значение тайм-аута выполнения SQL операторов в указанных единицах измерения времени. Если единица измерения времени не указано, то по умолчанию значение тайм-аута измеряется в секундах.

Данный SQL оператор работает вне механизма управления транзакциями и вступают в силу немедленно.

```
SET STATEMENT TIMEOUT 3 MINUTE;
```

SET STATISTICS

Оператор позволяет улучшить селективность (избирательность) указанного индекса. Улучшенная селективность увеличивает скорость выборки и упорядочения данных, при которых используется данный индекс. Синтаксис оператора:

Листинг Ж.112. Синтаксис оператора SET STATISTICS INDEX

```
SET STATISTICS INDEX <имя индекса>;
```

Улучшение селективности всех индексов базы данных можно также получить, выполнив резервное копирование и восстановление базы данных. См. документ «Руководство администратора».

Только владелец таблицы, для которой был создан индекс, администратор и пользователь с ролью `ALTER ANY TABLE` имеют привилегии на использование `SET STATISTICS INDEX`.

Подробно оператор описан в главе 14.

См. также операторы CREATE INDEX, DROP INDEX, ALTER INDEX.

SET TIME ZONE

Оператор позволяет изменить часовой пояс сеанса (текущего подключения).

Листинг Ж.113. Синтаксис оператора SET TIME ZONE

```
SET TIME ZONE {'<часовой пояс>' | LOCAL}

<часовой пояс> ::= <регион часового пояса>
                 | [+/-] <смещения часов относительно GMT> [: <смещения минут
относительно GMT>]
```

Данный SQL оператор работает вне механизма управления транзакциями и вступают в силу немедленно.

```
set time zone '-02:00';
set time zone 'America/Sao_Paulo';
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
set time zone local;
```

SET TIME ZONE BIND

Оператор изменяет привязку часового пояса сеанса для совместимости со старыми клиентами.

Листинг Ж.114. Синтаксис оператора SET TIME ZONE BIND

```
SET TIME ZONE BIND { NATIVE | LEGACY }
```

Значение по умолчанию - NATIVE, что означает, что выражения TIME WITH TIME ZONE и TIMESTAMP WITH TIME ZONE возвращаются с новыми типами данных для клиента.

Старые клиенты могут не понимать новые типы данных, поэтому можно определить привязку к LEGACY, и выражения будут возвращены как TIME WITHOUT TIME ZONE и TIMESTAMP WITHOUT TIME ZONE с соответствующим преобразованием.

Конфигурация привязки применима также к входным параметрам.

SET TRANSACTION

Оператор запускает на выполнение транзакцию с указанным именем и с заданными характеристиками. Синтаксис оператора:

Листинг Ж.115. Синтаксис оператора SET TRANSACTION

```
SET TRANSACTION
[NAME <имя транзакции>]
[READ WRITE | READ ONLY]
[WAIT [LOCK TIMEOUT <кол-во секунд>] | NO WAIT]
[[ISOLATION LEVEL] <уровень изоляции>]
[NO AUTO UNDO]
[IGNORE LIMBO]
[AUTO COMMIT]
[AUTO RELEASE TEMP BLOBID]
[RESERVING <список таблиц для резервирования> | USING <хендл БД> [,<хендл БД>]];

<уровень изоляции> ::=
            SNAPSHOT [TABLE [STABILITY]]
          | SNAPSHOT AT NUMBER <номер снимка транзакции>
          | READ COMMITTED [{[NO] RECORD_VERSION | READ CONSISTENCY}]

<список таблиц для резервирования> ::=
            <имя таблицы> [, <имя таблицы> ...]
            [FOR [SHARED | PROTECTED] {READ | WRITE}]
            [, <список таблиц для резервирования>] ...
```

Все предложения в операторе SET TRANSACTION являются необязательными. Если в операторе не задано никакого предложения, то предполагается запуск транзакции со значениями по умолчанию:

```
SET TRANSACTION
READ WRITE
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
  WAIT
  ISOLATION LEVEL SNAPSHOT;
```

Основными характеристиками любой транзакции являются: режим доступа к данным (`READ WRITE, READ ONLY`), режим разрешения блокировок (`WAIT, NO WAIT`) с возможным дополнительным уточнением (`LOCK TIMEOUT`), уровень изоляции (`ISOLATION LEVEL`) и средства резервирования или освобождения таблиц (предложение `RESERVING`).

Подробно оператор, работа с транзакциями, использование вложенных транзакций описаны в главе 5.

См. также операторы SAVEPOINT, ROLLBACK, COMMIT, RELEASE SAVEPOINT.

SET TRUSTED ROLE

Оператор `SET TRUSTED ROLE` включает доступ доверенной роли, при условии, что `CURRENT_USER` получен с помощью доверительной аутентификации и роль доступна.

Листинг Ж.116. Синтаксис оператора `SET TRUSTED ROLE`

```
SET TRUSTED ROLE;
```

Идея отдельной команды `SET TRUSTED ROLE` состоит в том, чтобы при подключении доверенного пользователя не указывать никакой дополнительной информации о роли, `SET TRUSTED ROLE` делает доверенную роль (если таковая существует) текущей ролью без дополнительной деятельности, связанной с установкой параметров DBP.

Доверенная роль это не специальный тип роли, ей может быть любая роль, созданная с помощью оператора `CREATE ROLE` или предопределённая системная роль `RDB$ADMIN`. Она становится доверенной ролью для подключения, когда подсистема отображения объектов безопасности (security objects mapping subsystem) находит соответствие между результатом аутентификации, полученным от плагина и локальным или глобальным отображением (mapping) для текущей базы данных. Роль даже может быть той, которая не предоставлена явно этому доверенному пользователю.

Доверенная роль не назначается при подключении по умолчанию. Можно изменить это поведение, используя соответствующий плагин аутентификации и команды `CREATE/ALTER MAPPING`.

См. также оператор SET ROLE.

SIMILAR TO

Оператор `SIMILAR TO` проверяет соответствие строки шаблону регулярного выражения SQL. Для успешного выполнения шаблон должен соответствовать всей строке — соответствие подстроки не достаточно. Если один из операндов имеет значение `NULL`, то и результат будет `NULL`. В противном случае результат является `TRUE` или `FALSE`. Предикат может быть использован в любом контексте, который принимает булевы (логические) выражения, такие как предложения `WHERE`, ограничения `CHECK` и PSQL-оператор `IF()`.

Синтаксис оператора `SIMILAR TO`:

Листинг Ж.117. Синтаксис оператора `SIMILAR TO`

```
<строка> [ NOT ] SIMILAR TO <регулярное выражение> [ESCAPE <символ
экранирования>]

<регулярное выражение> := <строковый элемент> [<квантификатор>] [['|'] <строковый
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
элемент> [ <квантификатор>] ...]

<квантификатор> ::= ? | * | + | '{' m [,[n]] '}'

<строковый элемент> ::= { <экранированный символ> | <обычный символ>}
                        | %
                        | <класс символов>
                        | ( <регулярное выражение> )

<экранированный символ> ::= <символ экранирования> <специальный символ>
                          | <символ экранирования> <символ экранирования>

<специальный символ> ::= '[' | ']' | '(' | ')' | '|' | '^' | '-' | '+' | '*' | '%' |
'_' | '?' | '{' | '}'

<обычный символ> ::= любой символ за исключением <специальный символ> и не
эквивалентный <символ экранирования> (если задан)

<класс символов> ::= '_'
                   | '[' <элемент класса> ... ']'
                   | '[^' <не элемент класса> ... ']'
                   | '[' <элемент класса> ... '^' <не элемент класса> ... ']'

<элемент класса>, <не элемент класса> ::= {<экранированный символ>|<обычный символ>}
                                        | <диапазон>
                                        | <предопределенные классы>

<диапазон> ::= { <экранированный символ> | <обычный символ>} -
               { <экранированный символ> | <обычный символ>}

<предопределенные классы> ::= '[:'ALPHA':]' | '[:'UPPER':]' | '[:'LOWER':]'
                            | '[:'DIGIT':]' | '[:'ALNUM':]' | '[:'SPACE':]'
                            | '[:'WHITESPACE':]'
```

Подробно оператор с примерами описан в главе 2.

SUSPEND

Начиная с версии РЕД Базы Данных 5.0.3 оператор недоступен.

Оператор SUSPEND позволяет заблокировать сеансы пользователей и ролей.

Листинг Ж.118. Синтаксис оператора SUSPEND

```
SUSPEND { [USER <список пользователей>] | [ROLE <список ролей>] }
[DISCONNECT] | [PERMANENT];

<список пользователей> ::= '<имя пользователя 1>' [, '<имя пользователя 2>' ...]

<список ролей> ::= '<имя роли 1>' [, '<имя роли 2>' ...]
```

Заблокировать сеанс для списка пользователей или ролей может только SYSDBA, владелец базы данных, пользователь с ролью RDB$ADMIN. Заблокировать собственный сеанс может любой пользователь.

Заблокированный пользователь будет получать ошибку "Connection suspended" на все запросы, кроме повторного подключения к базе данных, отключения от базы данных или создания нового подключения к базе данных.

```
SUSPEND USER TEST_USER;
```

UPDATE

Оператор UPDATE используется для изменения значений столбцов существующих строк таблицы или представления (таблиц, лежащих в основе представления). Синтаксис оператора:

Листинг Ж.119. Синтаксис оператора UPDATE

```
UPDATE {<имя таблицы> | <имя представления>} [[AS] <псевдоним>]
SET <имя столбца> = <значение>|DEFAULT [,<имя столбца> = <значение>|DEFAULT...]
[WHERE { <условие поиска> | CURRENT OF <имя курсора>}]
[PLAN <план>]
[ORDER BY <упорядочиваемый элемент> [, <упорядочиваемый элемент> ... ]]
[ROWS <m> [TO <n>]]
[SKIP LOCKED]
[RETURNING { <возвращаемые значения> | [{OLD. | NEW.}]* }
   [INTO [:]<имя переменной> [, [:]<имя переменной> ...]] ];

<значение> ::= { <литерал>
               | <выражение>
               | <встроенная функция>
               | <UDF> [(<параметр> [, <параметр> ...])]
               | NEXT VALUE FOR <имя генератора>
               | (<выбор одного>) }

<упорядочиваемый элемент> ::=
   {<имя столбца>|<псевдоним столбца>|<номер столбца>|<произвольное выражение>}
   [COLLATE <порядок сортировки>]
   [ASC[ENDING] | DESC[ENDING]]
   [NULLS {FIRST | LAST}]

<возвращаемые значения> ::= <имя столбца> [[AS] <алиас>] [,<имя столбца> [[AS] <алиас>
]]...
```

Изменять данные в таблице может ее владелец, пользователь SYSDBA, пользователь операционной системы root (Linux), trusted user (Windows), а также пользователь, которому предоставлено право изменять отдельные (указанные в операторе) столбцы таблицы оператором GRANT UPDATE — см. документ «Руководство администратора». Если в операторе изменение ключевых столбцов (столбцов, входящих в состав первичного или уникального ключа) таблицы влечет внесение изменений в строки дочерней таблицы, то и к этой таблице пользователь должен иметь соответствующие полномочия.

- Предложение SET задает список выполняемых изменений: указывается имя изменяемого столбца и после знака равенства новое значение;
- Предложение WHERE определяет множество строк, к которым будет применяться операция изменения данных;

- Ключевое слово PLAN задает план выборки данных:
- Предложение ORDER BY задает упорядочение результатов выборки:
- Предложение SKIP LOCKED позволяет пропускать записи, заблокированные другими транзакциями, вместо того чтобы ждать их разблокирования или выдавать ошибку конфликта обновления;
- Необязательное предложение RETURNING указывает, что оператор возвращает значения заданных столбцов изменяемой строки.

Подробнее об операторе изменения данных см. в главе 11.

См. также операторы INSERT, DELETE, UPDATE OR INSERT.

UPDATE OR INSERT

Оператор UPDATE OR INSERT позволяет изменить существующие данные или добавить новые, если в таблице нет строк, соответствующих некоторому условию. Синтаксис оператора:

Листинг Ж.120. Синтаксис оператора UPDATE OR INSERT

```
UPDATE OR INSERT INTO
   {<имя таблицы> | <имя представления>} [(<имя столбца> [, <имя столбца> ...] )]
VALUES (<значение> | DEFAULT [, <значение> | DEFAULT ...] )
[MATCHING (<имя столбца> [, <имя столбца>] ...)]
[RETURNING { <возвращаемые значения> | [{OLD. | NEW.}]* }
   [INTO [:]<имя переменной> [, [:]<имя переменной> ...]] ];

<возвращаемые значения> ::= <имя столбца> [[AS] <алиас>] [, <имя столбца> [[AS]
<алиас>]]...
```

Для выполнения оператора UPDATE OR INSERT пользователь должен иметь полномочия и UPDATE, и INSERT к таблице (представлению).

Оператор позволяет изменить значения отдельных столбцов в существующей строке или нескольких строках, если найдено соответствие, или добавить одну новую строку, если соответствия не найдено. В случае добавления новой строки в операторе должны быть заданы значения всех столбцов, входящих в состав первичного ключа.

- После ключевого слова INTO помещается имя таблицы или представления, к которому применяется оператор. В круглых скобках следует необязательный список имен столбцов таблицы (представления).
- Ключевое слово VALUES является обязательным. После него в скобках следует список значений, присваиваемых соответствующим столбцам.
- Если не задано ключевое слово MATCHING, то в списке столбцов должны присутствовать все столбцы, входящие в состав первичного ключа. Если в операторе указано ключевое слово MATCHING и после него список имен столбцов, то поиск соответствующих строк осуществляется по этим столбцам.
- Предложение RETURNING возвращает значения указанных столбцов измененной или добавленной строки.

Подробнее об операторе см. в главе 11.

См. также операторы INSERT, DELETE, UPDATE, конструкцию NEXT VALUE FOR.

Приложение 3 Встроенные функции

В этом разделе описаны встроенные функции SQL. Функции можно разделить на три типа — скалярные, агрегатные и оконные (аналитические) функции.

Скалярные функции получают параметры и возвращают ровно одно значение. Параметром может быть литерал, имя столбца таблицы, предварительно определенный литерал и т.д.

Агрегатные функции выполняют вычисление на наборе значений и возвращают одиночное значение.

Оконные функции (также известные как аналитические функции) являются своего рода агрегатными функциями, не уменьшающими степень детализации. При этом агрегированные данные выводятся вместе с неагрегированными.

3.1 Скалярные функции

3.1.1 Функции для работы с контекстными переменными

RDB$GET_CONTEXT

Функция для работы с контекстными переменными. Функция возвращает значение типа `VARCHAR(N)` контекстной переменной одного из пространств имен:

- `SYSTEM` — предоставляет доступ к системным контекстным переменным. Эти переменные доступны только для чтения;
- `USER_SESSION` — предоставляет доступ к пользовательским контекстным переменным, заданным через функцию `RDB$SET_CONTEXT`. Переменные существуют в течение подключения;
- `USER_TRANSACTION` — предоставляет доступ к пользовательским контекстным переменным, заданным через функцию `RDB$SET_CONTEXT`. Переменные существуют в течение транзакции;
- `DDL_TRIGGER` — предоставляет доступ к системным контекстным переменным, доступным только во время выполнения DDL триггера. Эти переменные доступны только для чтения;
- `AUTHDATA` — предоставляет доступ к информации об аутентификации и ФИО пользователя.

Длина возвращаемого значения (N) определяется исходя из размера фактических данных. По умолчанию используется `VARCHAR(8192)`.

Функция `RDB$GET_CONTEXT` является заранее объявленной UDF, поэтому для вызова не требуется писать в базе объявление.

Листинг 3.1. Синтаксис функции `RDB$GET_CONTEXT`

```
RDB$GET_CONTEXT ('пространство имен', '<имя переменной>')
```

Пространство имен и имя переменной регистрочувствительны, должны быть непустыми строками и заключены в кавычки.

Пространства имен `USER_SESSION` и `USER_TRANSACTION` изначально пусты. Пользователь может создать и установить значение переменных в них функцией `RDB$SET_CONTEXT` и получить их значения из функции `RDB$GET_CONTEXT`.

Пространство имен `SYSTEM` доступно только для чтения. Оно содержит много предопределенных переменных, показанных в таблице 3.1.

Таблица 3.1 — Контекстные переменные в пространстве имён SYSTEM

Имя переменной	Описание
ENGINE_VERSION	Версия сервера (например, 5.0.0)
FULL_VERSION	Полная версия сборки СУБД (например, WI-V3.0.11.0 RedDatabase 5.0 SNAPSHOT.16 (9ec7320661241a96270a45741e9aae609d024ade))
EDITION	Установленная редакция СУБД РЕД База Данных: Open, Standard или Enterprise
DB_NAME	Полный путь к базе данных или, если подключение через путь запрещено, алиас
GLOBAL_CN	Последнее значение текущего глобального счётчика Commit Number
EXT_CONN_POOL_SIZE	Размер пула внешних соединений
EXT_CONN_POOL_IDLE_COUNT	Текущее количество неактивных соединений в пуле внешних соединений.
EXT_CONN_POOL_ACTIVE_COUNT	Текущее количество активных соединений в пуле внешних соединений.
EXT_CONN_POOL_LIFETIME	Время жизни неактивных соединений в пуле внешних соединений.
REPLICATING	Настроена ли репликация базы данных. Если настроена, то возвращает значение TRUE, если не настроена — FALSE.
REPLICATION_SEQUENCE	Текущее значение последовательности репликации (номер последнего сегмента, записанного в журнал репликации).
DB_GUID	GUID текущей базы данных.
DB_FILE_ID	Идентификатор текущей базы данных на уровне файловой системы.
REPLICA_MODE	Режим реплики базы данных: 'READ-ONLY', 'READ-WRITE' и NULL.
EDITION	Установленная редакция СУБД РЕД База Данных: Open, Standard или Enterprise.
SESSION_ID	Глобальная переменная CURRENT_CONNECTION
NETWORK_PROTOCOL	Протокол, используемый в соединении с базой данных: 'TCPv4', 'WNET', 'XNET' или NULL
WIRE_COMPRESSED	Используется ли сжатие сетевого трафика. Если используется сжатие сетевого трафика возвращает TRUE, если не используется — FALSE. Для встроенных соединений — возвращает NULL.
WIRE_ENCRYPTED	Используется ли шифрование сетевого трафика. Если используется шифрование сетевого трафика возвращает TRUE, если не используется — FALSE. Для встроенных соединений — возвращает NULL.
WIRE_CRYPT_PLUGIN	Если соединение зашифровано - возвращает имя текущего плагина, иначе NULL.
CLIENT_ADDRESS	Для TCPv4 — IP адрес, для XNET — локальный ID процесса. Для всех остальных протоколов переменная имеет значение NULL
CLIENT_HOST	Имя хоста удаленного клиента. Значение возвращается для всех поддерживаемых протоколов.
CLIENT_PID	PID процесса на клиентском компьютере.
CLIENT_PROCESS	Полный путь к клиентскому приложению, подключившемуся к базе данных. Позволяет не использовать системную таблицу MON$ATTACHMENTS (поле MON$REMOTE_PROCESS)

(разрыв таблицы)

(разрыв таблицы)

Имя переменной	Описание
CURRENT_USER	Глобальная переменная CURRENT_USER
CURRENT_ROLE	Глобальная переменная CURRENT_ROLE
SESSION_IDLE_TIMEOUT	Содержит текущее значение тайм-аут простоя соединения в секундах, который был установлен на уровне соединения, или ноль, если тайм-аут не был установлен.
STATEMENT_TIMEOUT	Содержит текущее значение тайм-аута выполнения оператора в милисекундах, который был установлен на уровне подключения, или ноль, если тайм-аут не был установлен.
CURRENT_ROLES	Действующие в данный момент роли пользователя
LDAP_ROLES	Роли пользователя, полученные из LDAP
LDAP_ROLES_DN	DN ролей пользователя, полученных из LDAP
EFFECTIVE_USER	Эффективный пользователь в текущий момент. Указывает пользователя с привилегиями которого в текущий момент времени выполняется процедура, функция или триггер.
SESSION_TIMEZONE	Часовой пояс текущего соединения.
TRANSACTION_ID	Глобальная переменная CURRENT_TRANSACTION
ISOLATION_LEVEL	Уровень изоляции текущей транзакции CURRENT_TRANSACTION: 'READ COMMITTED', 'SNAPSHOT' или 'CONSISTENCY'
LOCK_TIMEOUT	Время ожидания транзакцией высвобождения ресурса при блокировке (в секундах)
READ_ONLY	Является ли транзакция только для чтения. Если является, то значений TRUE, если нет — FALSE
SNAPSHOT_NUMBER	Номер моментального снимка базы данных: уровня транзакции (для транзакции SNAPSHOT или CONSISTENCY) или уровня запроса (для транзакции READ COMMITTED READ CONSISTENCY). NULL, если моментальный снимок не существует.
PAGES_ALLOCATED	Количество страниц, выделенных для базы данных.
PAGES_USED	Количество страниц, используемых базой данных
PAGES_FREE	Количество свободных страниц в базе данных.

Если запрошенная переменная существует в данном пространстве имен, то будет возвращено её значение в виде строки с длиной по умолчанию 8192 символа. Обращение к несуществующему пространству имён или несуществующей переменной в пространстве имен SYSTEM приведёт к ошибке. Если Вы опрашиваете несуществующую переменную в одном из пространств имен USER_SESSION и USER_TRANSACTION, функция вернёт NULL.

Использование пространства имён DDL_TRIGGER допустимо, только во время работы DDL триггера. Его использование также допустимо в хранимых процедурах и функциях, вызванных триггерами DDL.

Контекст DDL_TRIGGER работает как стек. Перед возбуждением DDL триггера, значения, относящиеся к выполняемой команде, помещаются в этот стек. После завершения работы триггера значения выталкиваются. Таким образом. В случае каскадных DDL операторов, когда каждая пользовательская DDL команда возбуждает DDL триггер, и этот триггер запускает другие DDL команды, с помощью EXECUTE STATEMENT, значения переменных в пространстве имён DDL_TRIGGER будут соответствовать команде, которая вызвала последний DDL триггер в стеке вызовов.

Пространство имен DDL_TRIGGER доступно только для чтения. Оно содержит много предопреде-

ленных переменных, показанных в таблице 3.2.

Таблица 3.2 — Контекстные переменные в пространстве имён DDL_TRIGGER

Имя переменной	Описание
EVENT_TYPE	Тип события (CREATE, ALTER, DROP)
OBJECT_TYPE	Тип объекта (TABLE, VIEW и др.)
DDL_EVENT	Имя события. DDL_EVENT = EVENT_TYPE || ' ' || OBJECT_TYPE
OBJECT_NAME	Имя объекта метаданных
OLD_OBJECT_NAME	Имя объекта метаданных до переименования
NEW_OBJECT_NAME	Имя объекта метаданных после переименования
SQL_TEXT	Текст SQL запроса

Пространство имен AUTHDATA доступно только для чтения. Оно содержит предопределенные переменные, показанные в таблице 3.3.

Таблица 3.3 — Контекстные переменные в пространстве имён AUTHDATA

Имя переменной	Описание
AUTH_TYPE	Тип аутентификации: SECURITY или LDAP
AUTH_PLUGIN	Плагин аутентификации: • Legacy_Auth; • Srp; • GostPassword; • Win_Sspi; • Gss; • Certificate.
USER_FIRST_NAME	Дополнительная информация: имя пользователя. При аутентификации через LDAP они считываются из атрибута пользователя "CN".
USER_MIDDLE_NAME	Дополнительная информация: отчество пользователя.При аутентификации через LDAP они считываются из атрибута пользователя "CN".
USER_LAST_NAME	Дополнительная информация: фамилия пользователя. При аутентификации через LDAP они считываются из атрибута пользователя "CN".
LDAP_SERVER	Адрес сервера.

См. также функцию RDB$SET_CONTEXT.

RDB$SET_CONTEXT

Функция для работы с контекстными переменными. Функция создает переменную, устанавливает ее значение или обнуляет в одном из используемых пользователями для записи пространстве имён: USER_SESSION, USER_TRANSACTION.

В рамках одного соединения может быть максимум 1000 переменных. Является заранее объявленной UDF, поэтому для вызова не требуется писать в базе объявление.

Листинг 3.2. Синтаксис функции RDB$SET_CONTEXT

```
RDB$SET_CONTEXT ('USER_SESSION'|'USER_TRANSACTION', '<имя переменной>', '<значение
переменной>' | NULL)
```

Параметр `<имя переменной>` — регистрочувствительная строка с максимальной длиной 80 символов. Параметр `<значение переменной>` — значение любого типа, приводимое к типу `VARCHAR(N)`. Длина строки (N) определяется на этапе подготовки запроса по фактически переданному аргументу. Если тип аргумента на этапе подготовки не определен (используется параметр), то по умолчанию используется тип `VARCHAR(8192)`.

Пространства имен `USER_SESSION` и `USER_TRANSACTION` изначально пусты. Пользователь может создать и установить значение переменных в них функцией `RDB$SET_CONTEXT` и получить их значения из функции `RDB$GET_CONTEXT`. Контекст `USER_SESSION` связан с текущим соединением. Переменные в `USER_TRANSACTION` существуют только в рамках транзакции, в которой они были созданы. Все переменные в этом пространстве имён сохраняются при `ROLLBACK RETAIN` или `ROLLBACK TO SAVEPOINT`, независимо от того, в какой точке во время выполнения транзакции они были установлены. При завершении транзакции (при её подтверждении или отмене) контекст и все переменные, созданные в ней, уничтожаются.

Функция возвращает только два значения типа `INTEGER`: 1 — если переменная уже существовала и 0 — если не существовала.

Для удаления переменной надо установить её значение в `NULL`. Если данное пространство имен не существует, то функция вернёт ошибку.

См. также функцию RDB$GET_CONTEXT.

3.1.2 Функции для работы с файлами

CREATE_FILE

Создает файл в директории, прописанной в `directories.conf`, и заполняет его `BLOB` данными.

Листинг 3.3. Синтаксис функции CREATE_FILE

```
CREATE_FILE(<псевдоним директории>, <имя файла>, <BLOB-данные>)
```

Перед вызовом этой функции нужно в `directories.conf` в секции `blobs` прописать алиас каталога (`<псевдоним директории>`), в котором будут храниться созданные файлы с BLOB данными, с указанием реального пути к нему. Если данной директории не существует, функция создаст ее.

Функция создает файл такого формата:

```
<псевдоним директории>/<дата>/<имя файла>-<рандом>.<расширение>,
```

где

- `<дата>` — текущая дата в формате `YYYYMMDD`
- `<имя файла>` — исходное имя файла
- `<расширение>` — исходное расширение файла
- `<рандом>` — 22 рандомных символа в кодировке `BASE64`

Данная строка возвращается в качестве результата.

Пример

В директории с псевдонимом `test_dir` (путь к которой прописан в `directories.conf` в секции `blobs`) создадим текстовый файл `text.txt` с `BLOB`-данными. Для этого выполним команду:

```
select CREATE_FILE('test_dir', 'text.txt', cast('Hello World!' as blob))
from rdb$database;
----------------------------------------
test_dir/20150708/text-UnceByjB8Nba1Bbo6+h9lS.txt
```

См. также функции READ_FILE, DELETE_FILE.

DELETE_FILE

Удаляет файл из директории, прописанной в `directories.conf` в секции `blobs`. Ее синтаксис:

Листинг 3.4. Синтаксис функции DELETE_FILE

```
DELETE_FILE(<файл с BLOB>)
```

Параметр `<файл с BLOB>` — это строка в формате: `<псевдоним директории>/<имя файла>`. Удаление выполняется в момент подтверждения транзакции.

```
select DELETE_FILE('test_dir/20150708/text-UnceByjB8Nba1Bbo6+h9lS.txt')
from rdb$database
```

См. также функции CREATE_FILE, READ_FILE.

READ_FILE

Функция читает файл из директории, прописанной в `directories.conf` в секции `blobs`, и возвращает данные типа `BLOB`. Ее синтаксис:

Листинг 3.5. Синтаксис функции READ_FILE

```
READ_FILE(<файл с BLOB>)
```

Параметр `<файл с BLOB>` — это строка в формате: `<псевдоним директории>/<имя файла>`.

Перед вызовом этой функции нужно в `directories.conf` в секции `blobs` прописать алиас каталога (`<псевдоним директории>`), в котором хранятся файлы с BLOB данными, с указанием реального пути к нему. Директория должна существовать.

```
select READ_FILE('test_dir/20150708/text-UnceByjB8Nba1Bbo6+h9lS.txt')
from rdb$database
```

Функция не позволяет читать файлы в родительской директории алиаса, указанного в `directories.conf`.

См. также функции CREATE_FILE, DELETE_FILE.

3.1.3 Функции подсистемы безопасности

CHECK_DDL_RIGHTS

Системная функция проверки DDL прав на объекты. Ее синтаксис:

Листинг 3.6. Синтаксис функции CHECK_DDL_RIGHTS

```
CHECK_DDL_RIGHTS(<DDL-операция> ON <объект> WITH OWNER <имя пользователя>)
```

где:

- <DDL-операция>: ANY, CREATE, ALTER, DROP
- <объект>: TABLE, VIEW, PROCEDURE, FUNCTION, GENERATOR, EXCEPTION, SEQUENCE, DOMAIN, EXCEPTION, ROLE, SHADOW

Функция возвращает TRUE, если пользователь с именем <имя пользователя> имеет какие-либо DDL права на объекты типа <объект> (в эти права не входит право CREATE, рассматриваются права только на существующие объекты). Если указано слово ANY, то пользователь может создавать, удалять и модифицировать любой объект указанного типа.

См. также функцию CHECK_DML_RIGHTS.

CHECK_DML_RIGHTS

Системная функция проверки DML прав на объекты. Ее синтаксис:

Листинг 3.7. Синтаксис функции CHECK_DML_RIGHTS

```
CHECK_DML_RIGHTS ( <DML-операция> ON <объект> <имя объекта> [, <имя поля>])
```

где:

- <DML-операция>: ANY, INSERT, SELECT, UPDATE, DELETE, GRANT, REFERENCES, EXECUTE
- <объект>: TABLE, PROCEDURE, GENERATOR, VIEW
- <имя поля> — имя поля для типа объекта TABLE, VIEW

Функция возвращает TRUE, если пользователь имеет какие-либо DML права на объект (чтение, запись, выполнение и т. д.). Если указано слово ANY, то пользователь может выполнять все вышеперечисленные DML-операции над объектом указанного типа.

См. также функцию CHECK_DDL_RIGHTS.

RDB$ROLE_IN_USE

Функция RDB$ROLE_IN_USE возвращает используется ли роль текущим пользователем.

Листинг 3.8. Синтаксис функции RDB$ROLE_IN_USE

```
RDB$ROLE_IN_USE (<имя роли>)
```

Тип возвращаемого результата: BOOLEAN

Данная функция позволяет проверить использование любой роли: указанной явно (при входе в систему или изменённой с помощью оператора SET ROLE) и назначенной неявно (роли назначенные пользователю с использованием предложения DEFAULT).

RDB$SYSTEM_PRIVILEGE

Функция RDB$SYSTEM_PRIVILEGE возвращает используется ли системная привилегия текущим соединением.

Листинг 3.9. Синтаксис функции RDB$SYSTEM_PRIVILEGE

```
RDB$SYSTEM_PRIVILEGE (<системная привилегия>)
```

Тип возвращаемого результата: BOOLEAN

Список системных привилегий представлен в таблице.

MAKE_DBKEY

MAKE_DBKEY создает значение DBKEY, используя имя или идентификатор таблицы, номер записи и (не обязательно) номера страницы данных и страницы указателей.

Листинг 3.10. Синтаксис функции MAKE_DBKEY

```
MAKE_DBKEY (<таблица>, <номер записи> [, <номер страницы данных> [, <номер страницы
указателей>]])
```

Параметры функции имеют следующие значения:

- Таблица - имя (в одинарных кавычках) или идентификатор таблицы (значение поля RDB$RELATION_ID из таблицы RDB$RELATIONS);
- Номер записи - либо абсолютный номер записи (если аргументы <номер страницы данных> и <номер страницы указателей> отсутствуют), либо относительный номер записи (если присутствует аргумент <номер страницы данных>);
- Номер страницы данных - может быть либо абсолютным (если аргумент <номер страницы указателей> не задан), либо относительным (если указанный аргумент задан);
- Номер страницы указателя - логический номер страницы указателя в таблице.

Примеры работы функции MAKE_DBKEY:

- Выбор записи с использованием имени таблицы (обратите внимание, что имя таблицы указано в верхнем регистре)

```
select *
from rdb$relations
where rdb$db_key = make_dbkey('RDB$RELATIONS', 0);
```

- Выбор записи с использованием идентификатора таблицы

```
select *
from rdb$relations
where rdb$db_key = make_dbkey(6, 0);
```

- Выбор всех записей, физически находящихся на первой странице данных

```
select *
from rdb$relations
where rdb$db_key >= make_dbkey(6, 0, 0) and
      rdb$db_key < make_dbkey(6, 0, 1);
```

- Выбор всех записей, физически находящихся на первой странице данных шестой страницы указателей

```
select *
from SOMETABLE
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
  where rdb$db_key >= make_dbkey('SOMETABLE', 0, 0, 5) and
        rdb$db_key < make_dbkey('SOMETABLE', 0, 1, 5);
```

3.1.4 RDB$TRACE_MSG

Функция RDB$TRACE_MSG записывает указанное сообщение в лог-файл.

Для работы функции необходимо включить параметр log_message в fbtrace.conf.

Листинг 3.11. Синтаксис функции RDB$TRACE_MSG

```
RDB$TRACE_MSG('<сообщение>' [, <eol>])

<eol> ::= TRUE | FALSE
```

Сообщение представляет собой строковое значение, которое необходимо записать в лог-файл.

Параметр eol означает сброс буфера сообщений в лог. По умолчанию он включен (значение true), то есть каждое сообщение из буфера будет записано отдельным событием (MESSAGE) в лог-файл. Если eol выключен, то сообщение будет добавлено в буфер подключения. Сообщения из буфера будут записаны в лог, когда функция RDB$TRACE_MSG будет вызвана с включенным eol или, когда завершится текущий блок PSQL (execute block, функция, процедура).

Максимальный размер буфера составляет 1M6. Если при вызове RDB$TRACE_MSG содержимое буфера превышает 1M6, то будет получено сообщение об ошибке, а в лог будут записаны сообщения, добавленные в буфер до превышения лимита.

3.1.5 Функции для обработки ошибок

RDB$ERROR

Возвращает значение контекста активного исключения.

Листинг 3.12. Синтаксис функции RDB$ERROR

```
RDB$ERROR (<контекст>)

<контекст> ::= { GDSCODE | SQLCODE | SQLSTATE | EXCEPTION | MESSAGE }
```

Тип возвращаемого значения зависит от контекста:

- Если задан контекст EXCEPTION, то функция возвращает имя исключения, если активно исключение определённое пользователем, и NULL если активно одно из системных исключений. Для контекста EXCEPTION тип возвращаемого значения: VARCHAR(63) CHARACTER SET UTF8.
- Если задан контекст MESSAGE то функция возвращает интерпретированный текст активного исключения. Для контекста MESSAGE тип возвращаемого значения: VARCHAR(1024) CHARACTER SET UTF8.
- Если заданы контексты GDSCODE, SQLCODE или SQLSTATE, то функция возвращает значение соответствующей контекстной переменной.

Функция RDB$ERROR всегда возвращает NULL вне блока обработки ошибок WHEN ... DO.

3.1.6 Функции работы с транзакциями

RDB$GET_TRANSACTION_CN

Возвращает номер подтверждения (Commit Number) для заданной транзакции.

Листинг 3.13. Синтаксис функции RDB$GET_TRANSACTION_CN

```
RDB$GET_TRANSACTION_CN (<номер транзакции>)
```

Тип возвращаемого результата: BIGINT

Внутренние механизмы РЕД Базы Данных используют беззнаковое 8-байтное целое для Commit Number и беззнаковое 6-байтное целое для номера транзакции. Поэтому, не смотря на то, что язык SQL не имеет беззнаковых целых, а `RDB$GET_TRANSACTION_CN` возвращает знаковый `BIGINT`, невозможно увидеть отрицательный номер подтверждения, за исключением нескольких специальных значений, используемых для неподтверждённых транзакций.

Таким образом, числа возвращаемые `RDB$GET_TRANSACTION_CN` могут иметь следующие значения:

Значение	Описание
-2	Мёртвые транзакции (отмененные)
-1	Зависшие транзакции (в состоянии `limbo 2PC` транзакций)
0	Активные транзакции
1	Для транзакций подтверждённых до старта базы данных или с номером меньше чем OIT (Oldest Interesting Transaction)
>1	Транзакции подтверждённые после старта базы данных
NULL	Если номер транзакции равен `NULL` или больше чем `Next Transaction`

3.1.7 Математические функции

ABS

Функция возвращает абсолютное значение числового параметра с плавающей точкой или параметра, который может быть преобразован в число с плавающей точкой.

Листинг 3.14. Синтаксис функции ABS

```
ABS(<значение>)
```

Выходной параметр будет иметь тот же тип, что и у входного аргумента. Если входной параметр имеет значение `NULL`, то возвращается 0.

ACOS

Функция возвращает арккосинус числового параметра с плавающей точкой или параметра, который может быть преобразован в число с плавающей точкой.

Листинг 3.15. Синтаксис функции ACOS

```
ACOS(<значение>)
```

Входной параметр преобразуется в тип данных `DOUBLE PRECISION`. Может принимать значения от -1 до +1. Выходной параметр — угол в радианах. Возвращаемые значения находятся в диапазоне от 0 до числа π. Если входной параметр имеет значение `NULL`, то возвращается также пустое значение.

ACOSH

Функция возвращает обратный гиперболический косинус числового параметра с плавающей точкой или параметра, который может быть преобразован в число с плавающей точкой.

Листинг 3.16. Синтаксис функции ACOSH

```
ACOSH(<параметр>)
```

Входной параметр может принимать значения большие или равные 1. Возвращаемые значения находятся в диапазоне от 0 до $+\infty$. Если входной параметр имеет значение `NULL`, то возвращается также пустое значение.

ASIN

Функция возвращает арксинус числового параметра с плавающей точкой или параметра, который может быть преобразован в число с плавающей точкой.

Листинг 3.17. Синтаксис функции ASIN

```
ASIN(<значение>)
```

Входной параметр преобразуется в тип данных `DOUBLE PRECISION`. Может принимать значения от -1 до +1. Выходной параметр — угол в радианах. Возвращаемые значения находятся в диапазоне от 0 до числа π. Если входной параметр имеет значение `NULL`, то возвращается также пустое значение.

ASINH

Функция возвращает обратный гиперболический синус числового параметра с плавающей точкой или параметра, который может быть преобразован в число с плавающей точкой.

Листинг 3.18. Синтаксис функции ASINH

```
ASINH(<параметр>)
```

Входной параметр может принимать любые числовые значения. Возвращаемые значения находятся в диапазоне от $-\infty$ до $+\infty$. Если входной параметр имеет значение `NULL`, то возвращается также пустое значение.

ATAN

Функция возвращает арктангенс числового параметра с плавающей точкой или параметра, который может быть преобразован в число с плавающей точкой.

Листинг 3.19. Синтаксис функции ATAN

```
ATAN(<значение>)
```

Входной параметр преобразуется в тип данных DOUBLE PRECISION. Выходной параметр — угол в радианах. Возвращаемые значения находятся в диапазоне от $-\pi/2$ (для числа π можно использовать функцию PI) до числа $\pi/2$. Если входной параметр имеет значение NULL, то возвращается также пустое значение.

ATAN2

Функция возвращает арктангенс частного от деления первого числового параметра с плавающей точкой на второй числовой параметр.

Листинг 3.20. Синтаксис функции ATAN2

```
ATAN2(<значение 1>, <значение 2>)
```

Входные параметры преобразуются в тип данных DOUBLE PRECISION. Функция возвращает угол как отношение синуса к косинусу, аргументы, у которых задаются двумя параметрами, а знаки синуса и косинуса соответствуют знакам параметров. Возвращаемые значения находятся в диапазоне от $-\pi/2$ до $\pi/2$. Если любой из входных параметров или оба имеют значение NULL, то возвращается также пустое значение.

ATANH

Функция возвращает обратный гиперболический тангенс числового параметра с плавающей точкой или параметра, который может быть преобразован в число с плавающей точкой.

Листинг 3.21. Синтаксис функции ATANH

```
ATANH(<параметр>)
```

Входной параметр может принимать значения от -1 до 1. Возвращаемые значения находятся в диапазоне от $-\infty$ до $+\infty$. Если входной параметр имеет значение NULL, то возвращается также пустое значение.

CEIL|CEILING

Функция возвращает наименьшее целое число, превышающее значение входного параметра.

Листинг 3.22. Синтаксис функции CEILING

```
{CEILING | CEIL} (<значение>)
```

Любое число всегда можно записать в виде разницы целого N и положительного вещественного числа f таких, что <значение> = N-f и $0 \leq f < 1$. Результат CEIL - это число N.

Эта функция принимает в качестве аргумента любой числовой тип данных или любой нечисловой тип данных, который может быть неявно преобразован в числовой тип данных.

```
select CEIL(2.1), CEILING(-2.1) from rdb$database;
----------------------------------------
3, -2
```

См. также функцию FLOOR.

COS

Функция возвращает косинус заданного параметра, указанного в радианах. Возвращаемые значения находятся в диапазоне от –1 до + 1. Если входной параметр имеет значение NULL, то возвращается также пустое значение.

Листинг 3.23. Синтаксис функции COS

```
COS(<параметр>)
```

COSH

Функция возвращает гиперболический косинус заданного параметра, указанного в радианах. Если входной параметр имеет значение NULL, то возвращается также пустое значение.

Листинг 3.24. Синтаксис функции COSH

```
COSH(<параметр>)
```

COT

Функция возвращает значение 1, деленное на тангенс передаваемого функции значения. Другими словами - котангенс значения в радианах.

Листинг 3.25. Синтаксис функции COT

```
COT(<числовой параметр>)
```

Если параметр имеет значение NULL, то возвращается также пустое значение.

EXP

Функция возвращает число с плавающей точкой (типа данных DOUBLE PRECISION), которое получается при возведении экспоненты e (2,718281828459) в заданную параметром "целое число" степень. Синтаксис функции:

Листинг 3.26. Синтаксис функции EXP

```
EXP(<целое число>)
```

FLOOR

Функция FLOOR возвращает наибольшее целое число, меньшее или равное указанного числового выражения.

Листинг 3.27. Синтаксис функции FLOOR

```
FLOOR (<значение>)
```

Любое число всегда можно записать в виде суммы целого N и положительного вещественного числа f таких, что <значение> = N+f и $0 \leq f < 1$. Результат FLOOR — это число N.

Эта функция принимает в качестве аргумента любой числовой тип данных или любой нечисловой тип данных, который может быть неявно преобразован в числовой тип данных.

```
select FLOOR(2.1), FLOOR(-2.1) from rdb$database;
----------------------------------------
2, -3
```

См. также функцию CEIL|CEILING.

LN

Функция возвращает натуральный логарифм числа. Синтаксис:

Листинг 3.28. Синтаксис функции LN

```
LN(<числовой параметр>)
```

Входной параметр — положительное число типа данных DOUBLE PRECISION. Выходное значение также имеет тип данных DOUBLE PRECISION.

LOG

Функция возвращает логарифм числа по заданному основанию. Синтаксис:

Листинг 3.29. Синтаксис функции LOG

```
LOG(<числовой параметр 1>, <числовой параметр 2>)
```

Первый числовой параметр задает основание логарифма~ — положительное число типа данных DOUBLE PRECISION. Второй параметр — число, для которого вычисляется логарифм, тип данных параметра DOUBLE PRECISION. Выходное значение также имеет тип данных DOUBLE PRECISION.

См. также математические функции LN, LOG10.

LOG10

Функция возвращает десятичный логарифм числового параметра. Синтаксис:

Листинг 3.30. Синтаксис функции LOG10

```
LOG10(<числовой параметр>)
```

Числовой параметр — число, для которого вычисляется логарифм. Тип данных DOUBLE PRECISION. Выходное значение также имеет тип данных DOUBLE PRECISION.

См. также математические функции LN, LOG.

MOD

Функция возвращает остаток от деления первого целочисленного параметра на второй целочисленный параметр. Синтаксис:

Листинг 3.31. Синтаксис функции MOD

```
MOD(<числовой параметр 1>, <числовой параметр 2>)
```

Вещественные числа округляются до выполнения деления. Например, результатом `mod(7.5, 2.5)` будет 2, а не 0.

PI

Функция возвращает число "пи" (3.1459...) типа данных `DOUBLE PRECISION`.

POWER

Функция возводит число в степень. Синтаксис:

Листинг 3.32. Синтаксис функции `POWER`

```
POWER(<числовое значение 1>, <числовое значение 2>)
```

Функция возвращает значение первого числового параметра в степень, заданную вторым числовым параметром. Оба входных параметра и результат выполнения функции имеют тип данных `DOUBLE PRECISION`.

RAND

Функция возвращает случайное число типа данных `DOUBLE PRECISION` в диапазоне от 0 до 1.

ROUND

Функция выполняет округление числа. Синтаксис:

Листинг 3.33. Синтаксис функции `ROUND`

```
ROUND(<числовой параметр 1> [, <числовой параметр 2>])
```

Первое число округляется в соответствии с точностью, заданной вторым числовым параметром. Числовой параметр 2 задает количество требуемых знаков после десятичной точки в полученном числе. Если он имеет отрицательное значение, то происходит округление, при котором округляются указанные позиции целой части числа.

Если второй параметр не указан, то первое число округляется до ближайшего целого.

Если дробная часть равна 0.5, то округление до ближайшего большего целого числа для положительных чисел и до ближайшего меньшего для отрицательных чисел.

SIGN

Возвращает знак аргумента: -1, 0 или 1. Синтаксис:

Листинг 3.34. Синтаксис функции `SIGN`

```
SIGN(<числовой параметр>)
```

Функция возвращает +1, если число положительное, 0, если число равно нулю, и ˇ1, если число отрицательное.

SIN

Функция возвращает синус заданного параметра в радианах. Если входной параметр имеет значение NULL, то возвращается также пустое значение.

Листинг 3.35. Синтаксис функции SIN

```
SIN(<числовой параметр>)
```

SINH

Функция возвращает гиперболический синус заданного параметра в радианах. Если входной параметр имеет значение NULL, то возвращается также пустое значение.

Листинг 3.36. Синтаксис функции SINH

```
SINH(<числовой параметр>)
```

SQRT

Функция возвращает квадратный корень заданного параметра. Если входной параметр имеет значение NULL, то возвращается также пустое значение.

Листинг 3.37. Синтаксис функции SQRT

```
SQRT(<числовой параметр>)
```

Входной параметр и возвращаемое значение имею тип данных DOUBLE PRECISION.

TAN

Функция возвращает тангенс заданного параметра в радианах. Если входной параметр имеет значение NULL, то возвращается также пустое значение.

Листинг 3.38. Синтаксис функции TAN

```
TAN(<числовой параметр>)
```

TANH

Функция возвращает гиперболический тангенс заданного параметра. Если входной параметр имеет значение NULL, то возвращается также пустое значение.

Листинг 3.39. Синтаксис функции TANH

```
TANH(<числовой параметр>)
```

TRUNC

Функция возвращает число той же точности, обнуляя указанное количество знаков, заданное вторым параметром. Если числовой параметр имеет значение NULL, то возвращается также пустое значение.

Листинг 3.40. Синтаксис функции TRUNC

```
TRUNC(<числовой параметр> [, <масштаб>])
```

Масштаб задает уменьшение количества значащих цифр в числе. Может быть нулем (тогда возвращается целое число, дробные знаки отбрасываются), положительным числом (задается оставшееся количество знаков после десятичной точки) или отрицательным числом (тогда обнуляется указанное количество цифр в целой части числа).

```
select TRUNC(789.2225, 2) from rdb$database;
-- Результат: 789.2200 (а не 789.22)
select TRUNC(345.4, -2) from rdb$database;
-- Результат: 300.0 (а не 300)
select TRUNC(-163.41, 0) from rdb$database;
-- Результат: -163.00 (а не -163)
select TRUNC(-163.41) from rdb$database;
-- Результат: -163
```

3.1.8 Функции для работы со строками

ASCII_CHAR

Возвращает символ ASCII заданного числового параметра или параметра, который может быть преобразован в целое число. Если входным параметром задается дробное число, то происходит его банковское округление до целого числа. Результат возвращается в наборе символов NONE.

Листинг 3.41. Синтаксис функции ASCII_CHAR

```
ASCII_CHAR(<числовое значение>)
```

Входной параметр может принимать значения в диапазоне от 0 до 255, иначе выдается сообщение об арифметическом переполнении. Выходным параметром является символ. Если входной параметр имеет значение NULL, то возвращается пустое значение NULL.

См. также функцию ASCII_VAL.

ASCII_VAL

Возвращает число, соответствующее коду ASCII первого (единственного) символа заданного строкового параметра или параметра, который может быть преобразован в строку.

Листинг 3.42. Синтаксис функции ASCII_VAL

```
ASCII_VAL(<строка>)
```

Возвращается NULL, если входной параметр имеет значение NULL. Возвращается 0, если входной параметр является строкой, не содержащей ни одного символа (не пустым значением NULL, а строкой, не содержащей символов).

Входной параметр может принимать значение произвольного количества символов строкового типа данных. Выходным параметром является число.

См. также функцию ASCII_CHAR.

BIT_LENGTH

Функция BIT_LENGTH возвращает длину входной строки в битах. Для многобайтных наборов символов результат может быть в 8 раз больше, чем количество символов в "формальном" числе байт на символ, записанном в RDB$CHARACTER_SETS.

Листинг 3.43. Синтаксис функции BIT_LENGTH

```
BIT_LENGTH (<строка>)
```

С параметрами типа CHAR эта функция берет во внимание всю формальную строковую длину (например, объявленная длина поля или переменной). Если вы хотите получить "логическую" длину в битах, не считая пробелов, то перед передачей аргумента в BIT_LENGTH надо выполнить над ним операцию RIGHT TRIM.

Использование функции BIT_LENGTH:

```
select BIT_LENGTH ('Hello!') from rdb$database;
-- возвращает 48
select BIT_LENGTH (_ISO8859_1 'Grüß Di!') from rdb$database;
-- возвращает 80: ü и ß занимают 2 байта в ISO8859_1, остальные - 1
select BIT_LENGTH (CAST (_ISO8859_1 'Grüß di!' AS VARCHAR (24)
                   CHARACTER SET UTF8)) from rdb$database;
-- возвращает 112: ü и ß занимают по 4 байта в UTF8, остальные - 1
select BIT_LENGTH (CAST (_ISO8859_1 'Grüß di!' AS CHAR (24)
                   CHARACTER SET UTF8)) from rdb$database;
-- возвращает 224: размер всех 24 позиций CHAR - 1 байт и четыре из них 16-битные
```

См. также функции CHARACTER_LENGTH, OCTET_LENGTH.

CHARACTER_LENGTH

Функция CHARACTER_LENGTH (сокращенное название CHAR_LENGTH) возвращает количество символов, занимаемых входным параметром функции (константа, контекстная переменная, столбец таблицы). Для строки функция возвращает именно количество символов, а не байтов, отводимых под исходную строку. Если функция применяется к столбцу, который имеет набор символов, в котором для каждого символа используется более одного байта, то количество байтов этого столбца (функция OCTET_LENGTH) будет больше, чем количество символов.

Синтаксис функции CHARACTER_LENGTH:

Листинг 3.44. Синтаксис функции подсчета количества символов во входном параметре CHARACTER_LENGTH

```
{CHARACTER_LENGTH | CHAR_LENGTH} (<строка>)
```

См. также функции BIT_LENGTH, OCTET_LENGTH.

DAMLEV

Функция рассчитывает расстояние Дамерау — Левенштейна между двумя строками.

Листинг 3.45. Синтаксис функции DAMLEV

```
DAMLEV( <строка>, <строка> )
```

Возвращаемым значением является минимальное количество операций вставки, удаления, замены и транспозиции (перестановки двух соседних символов), необходимых для перевода одной строки в другую.

```
select DAMLEV('abcd','adcb') from rdb$database;
----------------------------------------
2
select DAMLEV('abcd','adcbe') from rdb$database;
----------------------------------------
3
```

LEFT

Возвращает указанные первые символы строки. Синтаксис:

Листинг 3.46. Синтаксис функции LEFT

```
LEFT(<строка>, <числовой параметр>)
```

Функция возвращает первые символы строки, указанные числовым параметром. Числовой параметр должен быть неотрицательным числом. Если он является дробным числом с фиксированной или плавающей точкой, то происходит его банковское округление до ближайшего целого числа. Если параметр 0, то возвращается пустая строка, но не NULL.

Если строка типа BLOB, результатом будет BLOB, в противном случае результатом будет VARCHAR(N), при этом N – будет равно длине строки. Если числовой параметр превысит длину текста, результатом будет исходный текст.

См. также функцию RIGHT.

LOWER

Переводит все буквы строки в нижний регистр. Синтаксис функции:

Листинг 3.47. Синтаксис функции LOWER

```
LOWER (<строка>)
```

Функция работает с латинскими буквами и с буквами кириллицы.

Выполнение функции

```
LOWER ('РОССИЯ')
```

вернет строку "россия".

Если исходное значение не содержит букв, то будет возвращаться само исходное значение. Если исходным параметром функции является столбец таблицы, то преобразование выполняется в соответствии с набором символов для этого столбца.

Точный результат зависит от набора символов входной строки. Например, для наборов символов NONE и ASCII только ASCII символы переводятся в нижний регистр; для OCTETS – вся входная строка возвращается без изменений.

Функция поддерживают тип данных BLOB.

См. также строковые функции UPPER, TRIM.

LPAD

Возвращает строку определенного вида. Синтаксис:

Листинг 3.48. Синтаксис функции LPAD

```
LPAD(<строка 1>, <числовой параметр> [, <строка      2>])
```

К строке, заданной параметром <строка 1>, в самое начало добавляется строка, заданная параметром <строка 2>, в том случае, если числовой параметр больше размера исходной строки.

Числовой параметр — неотрицательное целое число, не превышающее 32765. Если параметр имеет значение 0, то возвращается пустая строка (не NULL). Если значение числового параметра не превышает размера исходной строки (<строка 1>), то возвращаются первые заданные символы исходной строки.

Если опущена вторая необязательная строка (<строка 2>), то исходная строка дополняется слева пробелами до того размера, когда результирующая строка будет иметь длину, равную числовому параметру. Если при этом значение числового параметра меньше длины исходной строки, то происходит усечение этой строки справа до размера, заданного числовым параметром.

Если входная строка имеет тип BLOB, то результат также будет BLOB, в противном случае результат будет VARCHAR(<числовой параметр>).

```
select LPAD ('Hello' , 12) from rdb$database;                -- '       Hello'
select LPAD ('World' , 12, ',') from rdb$database;           -- ',,,,,,,World'
select LPAD ('Hello' , 12, '') from rdb$database;            -- 'Hello'
select LPAD ('World' , 12, 'abc') from rdb$database;         -- 'abcabcaWorld'
select LPAD ('Hello' , 12, 'abcdefghij') from rdb$database; -- 'abcdefgHello'
select LPAD ('World' , 2) from rdb$database;                 -- 'Wo'
select LPAD ('Hello' , 2, ',') from rdb$database;            -- 'He'
select LPAD ('World' , 2, '') from rdb$database;             -- 'Wo'
```

См. также строковую функцию RPAD.

OCTET_LENGTH

Возвращает количество байтов, занимаемых входным параметром функции. При работе с параметрами типа CHAR функция возвращает значение всей формальной строковой длины. Для того чтобы узнать "логическую" длину строки в байтах, то перед передачей аргумента функции следует применить RIGHT TRIM.

Листинг 3.49. Синтаксис функции OCTET_LENGTH

```
OCTET_LENGTH(<строка>)
```

Следует помнить, что не во всех наборах символов количество байт, занимаемых строкой, равно количеству символов.

```
select OCTET_LENGTH('Hello!') from rdb$database;
-- возвратит 6
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
select OCTET_LENGTH(_iso8859_1 'Grüß di!') from rdb$database;
-- возвратит 10: ü и ß занимают 2 байта в ISO8859_1
select OCTET_LENGTH(CAST(_iso8859_1 'Grüß di!' AS VARCHAR(24) CHARACTER SET utf8))
from rdb$database;
-- возвратит 14: ü и ß занимают 4 байта в UTF8
select OCTET_LENGTH(CAST(_iso8859_1 'Grüß di!' AS CHAR(24) CHARACTER SET utf8))
from rdb$database;
-- возвратит 28: всего 24 CHAR позиции, и четыре из них занимают 2 байта
```

См. также функции BIT_LENGTH, CHARACTER_LENGTH.

OVERLAY

Функция переписывает часть строки. Синтаксис:

Листинг 3.50. Синтаксис функции OVERLAY

```
OVERLAY(<строка 1> PLACING <строка 2>
FROM <начальная позиция> [FOR <количество заменяемых символов>])
```

Часть символов в первой строке заменяется на вторую строку. Символы заменяются, начиная с позиции, заданной после ключевого слова FROM. Количество заменяемых символов исходной строки задается после ключевого слова FOR. Если это ключевое слово отсутствует, то заменяется количество символов, равное количеству символов во второй строке.

Результат выполнения этой функции соответствует результату, полученному при использовании следующей операции конкатенации:

```
SUBSTRING(<строка 1> FROM 1 FOR <начальная позиция> - 1)
|| <строка 2>
|| SUBSTRING(<строка 1> FROM <начальная позиция>+<кол-во заменяемых символов>)
```

Особенности использования:

- Функция полностью поддерживает тестовые BLOB с любым набором символов и любой длины;
- Если входная строка имеет тип BLOB, то и результат будет иметь тип BLOB. В противном случае типом результата будет VARCHAR(n), где n является суммой длин входных строк; Как и во всех строковых функциях SQL параметр pos является определяющим;
- Если начальная позиция больше длины строки, то заменяющая строка помещается сразу после окончания первой строки;
- Если число символов от начальной позиции до конца строки меньше, чем длина второй строки (или, чем количество заменяемых символов, если задано), то первая строка усекается до значения начальная позиция и вторая строка помещается после неё;
- При нулевом количестве заменяемых символов (FOR 0) вторая строка просто вставляется в первую, начиная с указанной позиции;
- Если любой из параметров имеет значение NULL, то и результат будет NULL;
- Если параметры начальная позиция и количество заменяемых символов не являются целым числом, то используется банковское округление (до чётного): 0.5 становится 0, 1.5 становится 2, 2.5 становится 2, 3.5 становится 4 и т.д.

Использование функции OVERLAY:

```
select OVERLAY ('Goodbye' PLACING 'Hello' FROM 2) from rdb$database; -- 'GHelloe'
select OVERLAY ('Goodbye' PLACING 'Hello' FROM 5) from rdb$database; -- 'GoodHello'
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
select OVERLAY ('Goodbye' PLACING 'Hello' FROM 8) from rdb$database; -- 'GoodbyeHello'
select OVERLAY ('Goodbye' PLACING 'Hello' FROM 20) from rdb$database;-- 'GoodbyeHello'
select OVERLAY ('Goodbye' PLACING 'Hello' FROM 2 FOR 0) from rdb$database;
-- 'GHellooodbye'
select OVERLAY ('Goodbye' PLACING 'Hello' FROM 2 FOR 3) from rdb$database;
-- 'GHellobye'
select OVERLAY ('Goodbye' PLACING 'Hello' FROM 2 FOR 6) from rdb$database; -- 'GHello'
select OVERLAY ('Goodbye' PLACING 'Hello' FROM 2 FOR 9) from rdb$database; -- 'GHello'
select OVERLAY ('Goodbye' PLACING '' FROM 4) from rdb$database;           -- 'Goodbye'
select OVERLAY ('Goodbye' PLACING '' FROM 4 FOR 3) from rdb$database;      -- 'Gooe'
select OVERLAY ('Goodbye' PLACING '' FROM 4 FOR 20) from rdb$database;     -- 'Goo'
select OVERLAY ('' PLACING 'Hello' FROM 4) from rdb$database;              -- 'Hello'
select OVERLAY ('' PLACING 'Hello' FROM 4 FOR 0) from rdb$database;        -- 'Hello'
select OVERLAY ('' PLACING 'Hello' FROM 4 FOR 20) from rdb$database;       -- 'Hello'
```

См. также строковые функции UPPER, LOWER, TRIM, LPAD, SUBSTRING, REPLACE.

POSITION

Отыскивает позицию подстроки в исходной строке. Существует два варианта синтаксиса:

Листинг 3.51. Синтаксис функции POSITION

```
POSITION(<строка 1> IN <строка 2>)
POSITION(<строка 1>,<строка 2> [,<начальная позиция>[,<номер вхождения подстроки>]])
```

Функция возвращает целое число — позицию подстроки (строка 1) в исходной строке (строка 2). Если подстрока отсутствует в исходной строке, то функция возвращает 0. Третий аргумент (во втором варианте синтаксиса) задаёт позицию в строке, с которой начинается поиск подстроки, тем самым игнорируются любые вхождения подстроки в исходную строку до этой позиции. Четвёртый аргумент определяет, какое по счету вхождение подстроки нужно искать. Если аргумент не задан, то по умолчанию равен 1, то есть результатом будет позиция первого вхождения подстроки в строку.

REGEXP_SUBSTR

Строковая функция `REGEXP_SUBSTR` расширяет функциональность SUBSTRING, позволяя искать подстроку, которая соответствует регулярному выражению. Синтаксис функции выглядит следующим образом:

Листинг 3.52. Синтаксис функции REGEXP_SUBSTR

```
REGEXP_SUBSTR(<исходная строка>, <шаблон>[,<номер группы>, <номер вхождения>,
<параметр сравнения>, <позиция>])
```

Функция возвращает найденную подстроку или `NULL`, если поиск не дал результата. Тип возвращаемого значения зависит от типа входного параметра `<исходная строка>`. Если он имеет строковый тип (`CHAR, VARCHAR`), возвращается строка в той же кодировке. Если параметр — `BLOB`, возвращается также `BLOB` того же подтипа и в той же кодировке.

- `<исходная строка>` — строка для поиска в ней нужной подстроки.
- `<шаблон>` — регулярное выражение для поиска подстроки. О синтаксисе регулярного выражения будет рассказано далее.
- `<позиция>` - это порядковый номер символа исходной строки, с которого начнется поиск. По умолчанию 1, т.е. поиск с начала строки.

- `<номер вхождения>` — это положительно целое число показывает какое по счету вхождение подстроки (удовлетворяющей шаблону) в исходной строке нужно искать. Если этот параметр не задан или его значение меньше 1, он считается равным 1. При этом в качестве результата работы функции используется первая найденная подстрока.

  Если данный параметр больше 1, то ищутся следующие вхождения подстроки в исходную строку. Для этого на `N`-м этапе запускается поиск подстроки начиная с позиции `M+1` исходной строки, где `M` – конец подстроки, найденной на этапе `N-1`. Если на одном из этапов подстрока не найдена, возвращается `NULL`.
- `<параметр сравнения>` позволяет изменять поведение функции. Можно указать одно или более значений данного параметра, представленных в таблице:

Параметр	Описание
`I`	Нечувствительность к регистру. По умолчанию поиск чувствителен к регистру. Для корректной работы этого режима входная строка должна иметь правильно указанную кодировку и `COLLATE`, иначе ядро СУБД не сможет соотносить символы верхнего/нижнего регистра.
`G`	Включение "ленивого" (non-greedy) режима сопоставления. По умолчанию используется "жадный" режим, даже если используются квантификаторы `+?, *?, ?? и {n,}?`.
`M`	Режим множества строк (multi-line). Символы шаблона ^ и $ становятся спецсимволами, соответствующими началу и концу строки. Без этого режима символы ^ и $ не являются специальными (кроме использования ^ внутри [ ]).
`X`	Режим игнорирования пробельных символов (`free-spacing`). Все пробельные символы игнорируются в шаблоне. Чтобы при этом указать в шаблоне пробел, нужно его экранировать "", либо указать в квадратных скобках [ ]. Кроме того, в этом режиме символ решётки # начинает однострочный комментарий - игнорируется он сам и все символы после него до конца строки или регулярного выражения.
`S`	Режим единственной строки (single-line). Символ '.' будет соответствовать всем символам, в том числе переносу строки.
`T`	Обрезание пробелов справа в исходной строке и регулярном выражении. Необходим в случае передачи в функцию входных значений через механизм параметров в `EXECUTE STATEMENT`, так как в таком случае типы данных не сохраняются и значения дополняются пробелами справа.

- `<номер группы>` — это номер группы в шаблоне, которая будет использована в качестве результата. Если этот параметр не задан или его значение меньше 0, он считается равным 0, т.е. в качестве результата функции возвращается целиком найденная подстрока. Если параметр больше 0, то, если подстрока была найдена, из неё извлекается значение указанной в данном параметре группы. Если такой группы не существует или подстрока не найдена – возвращается `NULL`.

Примеры

```
select REGEXP_SUBSTR('abcd1234efgh5678', '([[:ALPHA:]]+[0-9]+)',1,1)
from rdb$database;                                              -- abcd1234
select REGEXP_SUBSTR('abcd1234efgh5678', '([[:ALPHA:]]+[0-9]+)',1,5)
from rdb$database;                                              -- efgh5678
select REGEXP_SUBSTR('abcd1234efgh5678', '([[:ALPHA:]]+[0-9]+)',1,5,'G')
from rdb$database;                                              -- efgh5
```

REPLACE

Отыскивает подстроку в исходной строке и заменяет на другую. Синтаксис:

Листинг 3.53. Синтаксис функции REPLACE

```
REPLACE(<исходная строка>, <отыскиваемая подстрока>, <строка замены>)
```

Функция выполняет замену в исходной строке всех найденных подстрок (отыскиваемая подстрока) на заданную третьим параметром строку замены.

Функция поддерживает текстовые BLOB любой длины и с любыми наборами символов. Если один из аргументов имеет тип BLOB, то результат будет иметь тип BLOB. В противном случае результат будет иметь тип VARCHAR(N), где N рассчитывается из длин всех параметров таким образом, что даже максимальное количество замен не будет вызывать переполнения поля.

Если отыскиваемая подстрока является пустой строкой, то возвращается исходная строка без изменений. Если заменяемая строка является пустой строкой, то все вхождения отыскиваемой подстроки удаляются из исходной.

Если любой из аргументов равен NULL, то результатом всегда будет NULL, даже если не было произведено ни одной замены

REVERSE

Функция возвращает значение строкового параметра, где символы располагаются в обратном порядке. Синтаксис:

Листинг 3.54. Синтаксис функции REVERSE

```
REVERSE(<строка>)
```

RIGHT

Возвращает указанные последние символы строки. Синтаксис:

Листинг 3.55. Синтаксис функции RIGHT

```
RIGHT(<строка>, <числовой параметр>)
```

Функция возвращает последние символы строки в количестве, указанном числовым параметром. Числовой параметр должен быть неотрицательным числом. Если он является дробным числом с фиксированной или плавающей точкой, то происходит его банковское округление до ближайшего целого числа. Если параметр 0, то возвращается пустая строка, но не NULL.

Функция поддерживает текстовые BLOB любой длины и с любыми наборами символов. Если строка типа BLOB, результатом будет BLOB, в противном случае результатом будет VARCHAR(N), при этом N – будет равно длине строки.

См. также функцию LEFT.

RPAD

Возвращает строку определенного вида. Синтаксис:

Листинг 3.56. Синтаксис функции RPAD

```
RPAD(<строка 1>, <числовой параметр> [, <строка 2>])
```

К строке, заданной параметром "строка 1", справа добавляется строка, заданная параметром "строка 2", в том случае, если числовой параметр больше размера исходной строки.

Числовой параметр — неотрицательное целое число, не превышающее 32765. Если параметр имеет значение 0, то возвращается пустая строка (не NULL). Если значение числового параметра не превышает размера исходной строки ("строка 1"), то возвращаются первые заданные символы исходной строки.

Если опущена вторая необязательная строка ("строка 2"), то исходная строка дополняется справа пробелами до того размера, когда результирующая строка будет иметь длину, равную числовому параметру. Если при этом значение числового параметра меньше длины исходной строки, то происходит усечение этой строки справа до размера, заданного числовым параметром.

Функция поддерживает текстовые BLOB любой длины и с любыми наборами символов. Если входная строка имеет тип BLOB, то результат также будет BLOB, в противном случае результат будет VARCHAR(<числовой параметр>).

```
select RPAD ('Hello' , 12) from rdb$database;                 -- 'Hello       '
select RPAD ('World' , 12, ',') from rdb$database;            -- 'World,,,,,,,'
select RPAD ('Hello' , 12, '') from rdb$database;             -- 'Hello'
select RPAD ('World' , 12, 'abc') from rdb$database;          -- 'Worldabcabca'
select RPAD ('Hello' , 12, 'abcdefghij') from rdb$database;   -- 'Helloabcdefg'
select RPAD ('World' , 2) from rdb$database;                  -- 'Wo'
select RPAD ('Hello' , 2, ',') from rdb$database;             -- 'He'
select RPAD ('World' , 2, '') from rdb$database;              -- 'Wo'
```

См. также строковую функцию LPAD.

SUBSTRING

Встроенная функция SUBSTRING возвращает подстроку исходной строки. Синтаксис функции:

Листинг 3.57. Синтаксис функции выделения подстроки SUBSTRING

```
SUBSTRING ( <строка> FROM <начальная позиция> [FOR <длина подстроки>]
          | <строка> SIMILAR <шаблон> ESCAPE <символ экранирования> )

<шаблон> ::= <шаблон: R1><символ экр-ия>"<шаблон: R2><символ экр-ия>"<шаблон: R3>
```

Здесь <строка> — исходное строка (строковый домен, столбец таблицы, входной или выходной параметр хранимой процедуры, строковый литерал, заключенный в апострофы, локальная переменная, используемая в хранимой процедуре или триггере).

Начальная позиция — номер позиции в строке, начиная с которой выделяется подстрока. Нумерация символов в строке начинается с единицы. Если начальная позиция подстроки превышает количество символов в строке, то будет выделена пустая подстрока — строка, содержащая ноль символов.

Длина подстроки — количество символов, которые выбираются в результирующую строку. Должно быть положительным числом. Если задать количество символов, которое выходит за границы исходной строки, то результат будет усечен до размера, соответствующего положению последнего символа исходной строки. При этом не будет выдано никаких диагностических сообщений. Если ключевое слово FOR не указано, то в подстроку помещаются все оставшиеся до конца исходной строки символы.

Функция полностью поддерживает двоичные и текстовые BLOB любой длины и с любым набором символов. Если исходная строка имеет тип BLOB, то и результат будет иметь тип BLOB. Для любых других типов результатом будет тип VARCHAR(N), где N всегда будет равен длине исходной строки.

Функция SUBSTRING с регулярным выражением возвращает часть строки, соответствующую шаблону в предложении SIMILAR. Если соответствия не найдено, то возвращается NULL.

Если любая из частей (R1, R2 или R3) регулярного выражения не является пустой строкой и не соответствует формату <шаблон>, будет возбуждено исключение.

Возвращаемое значение соответствует части R2 регулярного выражения. Для этого значения истинно выражение:

```
<строка> SIMILAR TO R1 || R2 || R3 ESCAPE <символ экранирования>
```

Если любой из входных параметров имеет значение NULL, то и результат тоже будет иметь значение NULL.

```
select SUBSTRING('Руководство ' FROM 5 FOR 3) from rdb$database;              -- вод
select SUBSTRING('abcdefg' SIMILAR 'a#"bcde#"fg' ESCAPE '#') from rdb$database;
-- bcde
select SUBSTRING('abcdefg' SIMILAR 'a#"%#"' ESCAPE '#') from rdb$database;  -- bcdefg
select SUBSTRING('abcdefg' SIMILAR '_#"%#"_' ESCAPE '#') from rdb$database;  -- bcdef
select SUBSTRING('abcdefg' SIMILAR '#"abc#"%' ESCAPE '#') from rdb$database;  -- abc
select SUBSTRING('abcdefg' SIMILAR '#"abc#"' ESCAPE '#') from rdb$database; -- <null>
```

Позиционный SUBSTRING

В простой позиционной форме (с FROM) эта функция возвращает подстроку, начинающуюся с указанной позиции (позиция первого сивола равна 1). Без аргумента FOR он возвращает все оставшиеся символы в строке. С использованием FOR возвращается длина подстроки или остаток строки, в зависимости от того, что короче.

Начальная позиция может быть меньше 1, тогда подстрока ведет себя так, как если бы строка имела дополнительные позиции 1 - начальная позиция перед фактическим первым символом в позиции 1. Значение длины подстроки считается от этого воображаемого начала строки, поэтому результирующая строка может быть короче указанной длины или даже пустой.

Функция полностью поддерживает двоичные и текстовые BLOB любой длины и с любым набором символов. Если входная строка имеет тип BLOB, то и результатом будет BLOB. Для любых других типов результатом будет тип VARCHAR.

Для входной строки, не являющейся BLOB, длина результата функции всегда будет равна длине входной строки, независимо от значений начальной позиции и длины подстроки.

Примеры

```
select SUBSTRING('abcdef' from 1 for 2) from rdb$database;
-- результат: 'ab'
select SUBSTRING('abcdef' from 2) from rdb$database;
-- результат: 'bcdef'
select SUBSTRING('abcdef' from 0 for 2) from rdb$database;
-- результат: 'a' не 'ab', потому что в позиции 0 нет "ничего"
select SUBSTRING('abcdef' from -5 for 2) from rdb$database;
-- результат: '', длина заканчивается до фактического начала строки
```

SUBSTRING по регулярному выражению

Функция SUBSTRING с регулярным выражением (с SIMILAR) возвращает часть строки, соответствующей шаблону регулярного выражения. Если соответствий не найдено, то возвращается NULL.

Шаблон SIMILAR формируется из трех шаблонов регулярных выражений: R1, R2 и R3. Полностью шаблон имеет форму R1 || '"<escape>"' || R2 || '"<escape>"' || R3, где <escape> - это escape-символ, определенный в предложении ESCAPE. R2 - это шаблон, который соответствует подстроке для извлечения и заключен в экранированные двойные кавычки ("<escape>", например, ""#"" с escape-символом '#'). R1 соответствует префиксу строки, а R3 - суффиксу строки. И R1, и R3 необязательны (они могут быть пустыми), но шаблон должен соответствовать всей строке. Другими словами, недостаточно указать шаблон, который находит только подстроку для извлечения.

Использование функции SUBSTRING с регулярными выражениями:

```
select SUBSTRING('abcabc' SIMILAR 'a#"bcab#"c' ESCAPE '#') from rdb$database; -- bcab
select SUBSTRING('abcabc' SIMILAR 'a#"%#"c' ESCAPE '#') from rdb$database;    -- bcab
select SUBSTRING('abcabc' SIMILAR '_#"%#"_' ESCAPE '#') from rdb$database;    -- bcab
select SUBSTRING('abcabc' SIMILAR '#"(abc)*#"' ESCAPE '#') from rdb$database;-- abcabc
select SUBSTRING('abcabc' SIMILAR '#"abc#"' ESCAPE '#') from rdb$database;  -- <null>
```

См. также строковые функции UPPER, LOWER, TRIM, LPAD, OVERLAY.

TRIM

Встроенная функция TRIM удаляет начальные и/или конечные указанные символы (по умолчанию пробелы) в исходной строке, передаваемой функции в виде входного параметра. Ее синтаксис:

Листинг 3.58. Синтаксис встроенной функции удаления символов в строке TRIM

```
<функция TRIM> ::= TRIM ( [[<спецификация удаления>][<удаляемые символы>] FROM]
<строка>)

<спецификация удаления> ::= LEADING | TRAILING | BOTH
```

Первый параметр — спецификация удаления — определяет, из какой части строки (начальной и/или конечной) будут удаляться указанные символы. Параметр может иметь следующие значения:

- LEADING — символы удаляются из начальной части строки.
- TRAILING — удаляются конечные символы строки.
- BOTH (значение по умолчанию) — символы одновременно удаляются как из начальной, так и из конечной части стоки.

Удаляемые символы — строка, содержащая произвольное количество символов. Эта строка заключается в апострофы. Если параметр опущен, предполагаются пробелы.

Строкой в функции может быть строковый столбец, домен, которому был задан строковый тип данных, строковый литерал, заключенный в апострофы, входной или выходной параметр хранимой процедуры, локальная переменная строкового типа данных.

Функция поддерживает тип BLOB. Если строка имеет тип BLOB, то и результат будет иметь тип BLOB. В противном случае результат будет иметь тип VARCHAR(n), где n является длиной строки.

Подстрока для удаления, если она, конечно, задана, не должна иметь длину больше, чем 32767 байта. Однако при повторениях подстроки в начале и/или конце строки общее число удаляемых байтов может быть гораздо больше.

Примеры. Результатом выполнения такой операции

```
select TRIM ('  Руководство ' || 'по SQL  ') from rdb$database;
```

будет строка: `Руководство по SQL`. Здесь по умолчанию убираются символы пробелов.

В результате выполнения следующей функции будут удалены только начальные символы "звездочка".

```
TRIM (LEADING '*' FROM '**********Руководство ' || 'по SQL*******')
```

Функция вернет строку `Руководство по SQL*******`.

В результате выполнения следующей функции будут удалены только конечные символы "звездочка".

```
TRIM (TRAILING '*' FROM '**********Руководство ' || 'по SQL*******')
```

Функция вернет строку `**********Руководство по SQL`.

Чтобы удалить как начальные, так и конечные символы "звездочка" из строки, нужно выполнить функцию:

```
TRIM (BOTH '*' FROM '**********Руководство ' || 'по SQL*******')
```

Ключевое слово `BOTH` можно не задавать. В этом случае удаляются указанные символы как с начала, так и с конца строки.

См. также функции BIT_LENGTH, CHARACTER_LENGTH, OCTET_LENGTH.

UPPER

Переводит все буквы строки в верхний регистр. Синтаксис функции:

Листинг 3.59. Синтаксис функции `UPPER`

```
UPPER(<строка>)
```

Функция работает не только с латинскими буквами, но и с буквами кириллицы.

Выполнение функции

```
UPPER ('россия')
```

вернет строку `"РОССИЯ"`.

Если исходное значение не содержит букв, то будет возвращаться само исходное значение. Если исходным параметром функции является столбец таблицы, то преобразование выполняется в соответствии с набором символов для этого столбца.

Точный результат зависит от набора символов входной строки. Например, для наборов символов `NONE` и `ASCII` только `ASCII` символы переводятся в верхний регистр; для `OCTETS` — вся входная строка возвращается без изменений.

Функция поддерживают тип данных `BLOB`.

См. также строковые функции LOWER, TRIM.

UNICODE_CHAR

Возвращает UNICODE символ для заданной кодовой точки.

Листинг 3.60. Синтаксис функции UNICODE_CHAR

```
UNICODE_CHAR( <числовое значение> )
```

Аргументом функции должна быть кодовая точка UTF-32 вне диапазона суррогатов верхней/-нижней границы (от 0xD800 до 0xDFFF). В противном случае будет ошибка.

```
select unicode_char(0x1F601) from rdb$database;
```

UNICODE_VAL

Возвращает число, соответствующее коду Unicode первого символа в указанной строке.

Листинг 3.61. Синтаксис функции UNICODE_VAL

```
UNICODE_VAL( <строка> )
```

```
select unicode_val(unicode_char(0x1F601)) from rdb$database;
```

3.1.9 Функции для работы с датой и временем

AT

Преобразует время или временную метку в указанный часовой пояс.

Листинг 3.62. Синтаксис функции UPPER

```
<выражение> AT {TIME ZONE '<часовой пояс>' | LOCAL}

<часовой пояс> ::=
         <регион часового пояса> | [+/-] <разница часов с GMT> [:<разница минут с GMT>
]
```

Если используется ключевое слово LOCAL, то преобразование происходит в часовой пояс сессии.

```
select time '12:00 GMT' at time zone '-03:00' from rdb$database;

select current_timestamp at time zone 'America/Sao_Paulo' from rdb$database;

select timestamp '2018-01-01 12:00 GMT' at local from rdb$database;
```

DATEADD

Возвращает значение типа данных DATE, TIME или TIMESTAMP в зависимости от типа данных входного параметра. Возвращаемое значение параметра увеличивается (уменьшается, если задано отрицательное значение параметра "целое число") на соответствующее количество секунд (миллисекунд, минут, часов, дней, месяцев, лет), заданных параметром "целое число". У функции есть два формата.

Листинг 3.63. Синтаксис функции DATEADD

```
DATEADD(<целое число> <элемент даты/времени> TO <входной параметр>)
DATEADD(<элемент даты/времени>, <целое число>, <входной параметр>)
```

Элемент даты/времени — это YEAR, MONTH, WEEK, DAY, WEEKDAY, YEARDAY, HOUR, MINUTE, SECOND, MILLISECOND. С типом данных, содержащим только время, не могут использоваться элементы, относящиеся к дате, с типом данных DATE не могут использоваться элементы времени. Для типа данных TIMESTAMP допустимы любые варианты.

Целое число в функции должно находиться в диапазоне от –2,147,483,648 до +2,147,483,647. Дробные знаки в числе отбрасываются без округления.

Функция возвращает значение, имеющее тот же тип данных, что и входной параметр, т.е. SMALLINT, INTEGER, BIGINT или NUMERIC.

При задании элементов, указывающих месяц, день, час, минуту, секунду или миллисекунду, происходит естественное изменение значений всех вышележащих элементов, составляющих дату и время. Например, вызов следующей функции вернет дату и время, приблизительно на 68 лет ранее текущей даты.

```
DATEADD(SECOND, -2147483648, CURRENT_TIMESTAMP)
```

Функция требует явного преобразования литералов к соответствующему типу данных.

См. также функции CAST, DATEDIFF, EXTRACT.

DATEDIFF

Возвращает целое число, задающее интервал в соответствии с указанным выделяемым элементом между двумя значениями типа данных DATE, TIME или TIMESTAMP. У функции есть два формата.

Листинг 3.64. Синтаксис функции DATEDIFF

```
DATEDIFF(<элемент даты/времени> FROM <входной пар-тр 1> TO <входной пар-тр 2>)
DATEDIFF(<элемент даты/времени>, <входной пар-тр 1>, <входной пар-тр 2>)
```

Элемент даты/времени — это YEAR, MONTH, WEEK, DAY, WEEKDAY, YEARDAY, HOUR, MINUTE, SECOND, MILLISECOND. С типом данных, содержащим только время, не могут использоваться элементы, относящиеся к дате, с типом данных DATE не могут использоваться элементы времени. Для типа данных TIMESTAMP допустимы любые варианты.

Функция возвращает количество интервалов, заданных элементом даты/времени (лет, месяцев, дней, часов, минут, секунд или миллисекунд) между двумя входными параметрами. Возвращается число со знаком: из второго параметра производится соответствующее вычитание элемента первого параметра.

Дата и время имеет естественную иерархическую структуру: год, месяц, день, час, минута, секунда, миллисекунда. При вычислении разности элементов одного уровня учитываются значения лишь этого или более высокого уровня. Элементы нижележащих уровней не учитываются.

Функция требует явного преобразования литералов к соответствующему типу данных.

Пример Чтобы определить, сколько лет осталось до 2050 года, нужно выполнить функцию:

```
DATEDIFF (YEAR, CURRENT_DATE, CAST('01.01.2050' AS DATE))
```

См. также функции DATEADD, EXTRACT, CAST.

EXTRACT

Функция для типов данных даты (DATE), времени (TIME) и даты/времени (TIMESTAMP) позволяет выделять различные элементы даты и времени. Синтаксис функции:

Листинг 3.65. Синтаксис функции EXTRACT

```
EXTRACT (<выделяемый элемент> FROM <дата>)
```

Исходным данным может быть столбец, домен (ключевое слово VALUE), параметр или внутренняя переменная хранимой процедуры или триггера.

Выделяемый элемент:

- YEAR — год: функция вернет целое число от 1 до 9999, ведущие нули отбрасываются,
- MONTH — месяц: вернет целое число от 1 до 12, ведущий ноль отбрасывается,
- DAY — день месяца: целое число от 1 до 31, ведущий ноль отбрасывается,
- HOUR — функция возвращает часы: целое число от 0 до 23,
- MINUTE — возвращаются минуты: целое число от 0 до 59,
- SECOND — секунды, включая десятитысячные доли секунды,
- MILLISECOND — возвращаются миллисекунды,
- WEEK — номер недели в году: целое число от 1 до 53,
- WEEKDAY — номер дня в неделе; 0 — воскресенье, 6 — суббота,
- YEARDAY — номер дня в году: число от 0 до 365. Первый день в году имеет номер 0,
- TIMEZONE_HOUR — функция возвращает смещение часов часового пояса: целое число от -23 до 23,
- TIMEZONE_MINUTE — функция возвращает смещение минут часового пояса: целое число от -59 до 59,
- QUARTER — квартал года, возвращает число от 1 до 4.

Выделять часы, минуты и секунды можно лишь в типах данных, содержащих время: TIME и TIMESTAMP. Выделение элементов даты возможно только для тех типов данных, которые содержат дату: DATE и TIMESTAMP.

В следующем операторе из переменной DATE_C типа DATE выделяются день, месяц и год. Полученные данные при помощи операции конкатенации приводятся к виду, принятому в нашей стране:

```
EXTRACT (DAY FROM DATE_C) || '.' ||
EXTRACT (MONTH FROM DATE_C) || '.' ||
EXTRACT (YEAR FROM DATE_C);
```

См. также функции DATEADD, DATEDIFF, CAST.

FIRST_DAY

Функция возвращает первый день года, месяца или недели для заданной даты.

Листинг 3.66. Синтаксис функции FIRST_DAY

```
FIRST_DAY( OF {YEAR | MONTH | WEEK | QUARTER} FROM <дата> )
```

Тип возвращаемого результата: DATE или TIMESTAMP

Первым днём недели считается воскресенье, как это возвращает функция EXTRACT с частью WEEKDAY.

Когда в качестве аргумента функции передаётся выражение типа TIMESTAMP, то возвращаемое значение сохраняет временную часть.

LAST_DAY

Функция возвращает последний день года, месяца или недели для заданной даты.

Листинг 3.67. Синтаксис функции FIRST_DAY

```
LAST_DAY( OF {YEAR | MONTH | WEEK | QUARTER} FROM <дата> )
```

Тип возвращаемого результата: DATE или TIMESTAMP

Последним днём недели считается суббота, как это возвращает функция EXTRACT с частью WEEKDAY.

Когда в качестве аргумента функции передаётся выражение типа TIMESTAMP, то возвращаемое значение сохраняет временную часть.

UTC_TIMESTAMP

Возвращает текущую дату и время по стандарту UTC в качестве значения в формате YYYY-MM-DD HH:MM:SS. Синтаксис:

Листинг 3.68. Синтаксис функции UTC_TIMESTAMP

```
UTC_TIMESTAMP
```

3.1.10 Функции для работы с типом DECFLOAT

COMPARE_DECFLOAT

Функция COMPARE_DECFLOAT сравнивает два значения типа DECFLOAT, которые могут быть одинаковыми, разными или неупорядоченными.

Листинг 3.69. Синтаксис функции COMPARE_DECFLOAT

```
COMPARE_DECFLOAT (<значение1>, <значение2>)
```

Тип возвращаемого результата: SMALLINT

Таблица 3.6 — Результирующие значения функции COMPARE_DECFLOAT

Результат	Описание
0	Значения равны
1	Первое значение меньше, чем второе
2	Первое значение больше, чем второе
3	Значения не упорядочены (одно или оба NAN / SNAN)

В отличие от операторов сравнения (<, >, = и др.) сравнение с помощью COMPARE_DECFLOAT является точным, т.е.

```
COMPARE_DECFLOAT(2.17, 2.170)
```

вернёт 2, а не 0

NORMALIZE_DECFLOAT

Функция NORMALIZE_DECFLOAT возвращает число в нормализованном виде. Это обозначает, что для любого ненулевого значения удаляются завершающие нули с соответствующей коррекцией экспоненты.

Листинг 3.70. Синтаксис функции NORMALIZE_DECFLOAT

```
NORMALIZE_DECFLOAT (<значение>)
```

Тип возвращаемого результата: DECFLOAT

```
select NORMALIZE_DECFLOAT(12.00) from rdb$database;-- 12
select NORMALIZE_DECFLOAT(120) from rdb$database;-- 1.2E+2
```

QUANTIZE

Функция QUANTIZE возвращает значение первого аргумента масштабированным с использованием второго значения в качестве шаблона.

Листинг 3.71. Синтаксис функции QUANTIZE

```
QUANTIZE (<значение>, <шаблон>)
```

Тип возвращаемого результата: DECFLOAT

Функция QUANTIZE возвращает значение DECFLOAT, равное по значению (за исключением любого округления) и знаку <значение>, а также экспоненте, равной по значению экспоненте <шаблон>. Функцию QUANTIZE можно использовать для реализации округления с точностью до нужного знака, например, округление до ближайшего цента с использованием установленного режима округления DECFLOAT.

Для значения шаблона не никаких ограничений, тем не менее при использовании SNaN функция выдаст исключение, при использовании NULL результатом будет NULL и т.д.

```
select v, pic, quantize(v, pic) from examples;
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
V           PIC         QUANTIZE
==========  ==========  ==========
3.16        0.001       3.160
3.16        0.01        3.16
3.16        0.1         3.2
3.16        1           3
3.16        1E+1        0E+1
-0.1        1           -0
0           1E+5        0E+5
316         0.1         316.0
316         1           316
316         1E+1        3.2E+2
316         1E+2        3E+2
```

TOTALORDER

Функция TOTALORDER сравнивает два значения типа DECFLOAT, включая специальные значения. Сравнение является точным.

Листинг 3.72. Синтаксис функции TOTALORDER

```
TOTALORDER (<значение1>, <значение2>)
```

Тип возвращаемого результата: SMALLINT

Таблица 3.7 — Результирующие значения функции TOTALORDER

Результат	Описание
-1	Первое значение меньше второго
0	Значения равны
1	Первое значение больше второго

Значения DEFLOAT в следующем виде:

```
-nan < -snan < -inf < -0.1 < -0.10 < -0 < 0 < 0.10 < 0.1 < inf < snan < nan
```

3.1.11 Кодирование и декодирование бинарных данных

BASE64_ENCODE

Функция BASE64_ENCODE кодирует входные данные в представлении BASE64. Функция может работать как с символьной строкой, так и с BLOB.

Листинг 3.73. Синтаксис функции BASE64_ENCODE

```
BASE64_ENCODE (<двоичные данные>)
```

Тип возвращаемого результата: BLOB или VARCHAR

BASE64_DECODE

Функция BASE64_DECODE декодирует входные данные из представления BASE64. Функция может работать как с символьной строкой, так и с BLOB.

Листинг 3.74. Синтаксис функции BASE64_DECODE

```
BASE64_ENCODE (<данные в base64>)
```

Тип возвращаемого результата: BLOB или VARCHAR

HEX_ENCODE

Функция HEX_ENCODE кодирует двоичные данные в шестнадцатеричное представлении. Функция может работать как с символьной строкой, так и с BLOB.

Листинг 3.75. Синтаксис функции HEX_ENCODE

```
HEX_ENCODE (<двоичные данные>)
```

Тип возвращаемого результата: BLOB или VARCHAR

HEX_DECODE

Функция HEX_DECODE декодирует данные в шестнадцатеричном представлении в двоичные данные. Функция может работать как с символьной строкой, так и с BLOB.

Листинг 3.76. Синтаксис функции HEX_DECODE

```
HEX_DECODE (<16-ричные данные>)
```

Тип возвращаемого результата: BLOB или VARCHAR

BLOB_APPEND

Оператор || с BLOB-аргументами создает временный BLOB для каждой пары аргументов, содержащих BLOB. Это может привести к чрезмерному потреблению памяти и увеличению файла базы данных. Функция BLOB_APPEND предназначена для объединения BLOB без создания промежуточных объектов.

Чтобы достичь этого, результирующий BLOB остается открытым для записи, а не закрывается сразу после заполнения данными. Данные в такой BLOB можно добавлять столько раз, сколько потребуется. Сервер помечает такой BLOB внутренним флагом BLB_close_on_read и закрывает его при необходимости.

Листинг 3.77. Синтаксис функции BLOB_APPEND

```
BLOB_APPEND( <значение> [, <значение>, ... <значение> ] )
```

Входные параметры:

- В зависимости от значения первого аргумента возможны следующие варианты поведения:
  - NULL — создается новый BLOB (незакрытый, с флагом BLB_close_on_read).

  - постоянный BLOB (из таблицы) или временный BLOB, который уже был закрыт — создается новый BLOB (незакрытый, с флагом BLB_close_on_read), его содержимое копируется из первого аргумента.
  - временный незакрытый BLOB — он будет использоваться в дальнейшем.
  - другие типы данных преобразуются в строку, создается новый BLOB (незакрытый, с флагом BLB_close_on_read), его содержимое копируется из этой строки.
- Другие аргументы могут быть любого типа, для них определено следующее поведение:
  - значения NULL игнорируются.
  - не BLOB преобразуются в строки и добавляются к результату.
  - BLOB при необходимости переводятся в кодировку первого аргумента, и их содержимое добавляется к результату.

Функция BLOB_APPEND возвращает временный незакрытый BLOB с флагом BLB_close_on_read. Это либо новый BLOB, либо тот, который указан в качестве первого аргумента. Таким образом, серия операций типа `blob = BLOB_APPEND (blob, ...)` приведет к созданию не более одного BLOB (если только вы не попытаетесь добавить BLOB к самому себе). Этот BLOB будет закрыт, когда клиент прочитает его, назначит его таблице или использует в других выражениях, требующих чтения содержимого.

Проверка BLOB на наличие значения NULL с помощью оператора IS [NOT] NULL не считывает его, и поэтому BLOB не будет закрыт после такой проверки.

Используйте функции LIST или BLOB_APPEND для объединения BLOB. Это уменьшает потребление памяти и дисковый ввод-вывод, а также предотвращает рост базы данных из-за создания большого количества временных BLOB.

```
execute block
  returns (b blob sub_type text)
as
begin
  -- создает новый временный незакрытый BLOB
  -- записывает в него строку из второго аргумента
  b = blob_append(null, 'Hello ');

  -- добавляет две строки во временный BLOB, не закрывая его
  b = blob_append(b, 'World', '!');

  -- сравнение BLOB со строкой приведет к его закрытию, потому что BLOB должен быть
прочитан
  if (b = 'Hello World!') then
  begin
      ...
  end

  --создает временный закрытый BLOB, добавляя к нему строку
  b = b || 'Close';
  suspend;
end!
```

3.1.12 Хэш функции

HASH

Функция возвращает хэш входного значения, используя для этого встроенный алгоритм хэширования. Если входное значение не является строковым или двоичным, то перед хэшированием оно преобразуется в строку.

Листинг 3.78. Синтаксис функции HASH

```
HASH (<значение> [USING <алгоритм>])

<алгоритм> ::= CRC32
```

- `Значение` - Выражение или значение любого типа. Нестроковые и небинарные типы будут преобразованы в строку. Функция поддерживает тип данных `BLOB`.
- `Алгоритм` - Алгоритм хэширования, который нужно применить.

Необязательное предложение `USING` указывает, какой алгоритм хэширования нужно применить. Если алгоритм не указан, то по умолчанию используется `PJW`. При использовании алгоритма `PJW` функция вернёт значение типа `BIGINT`.

При указании `CRC32` будет использован полином `0x04C11DB7`. В этом случае функция вернёт значение `INTEGER`.

Примеры использования функции `HASH`:

1. Хэширование с использованием алгоритма `CRC32`:

```
SELECT HASH(X USING CRC32) FROM Y;
```

2. Хэширование с использованием алгоритма `PJW`:

```
SELECT HASH(X) FROM Y;
```

См. также функцию HASH_CP.

CRYPT_HASH

Функция возвращает криптографический хэш для входного значения. Если входное значение не является строковым или двоичным, то перед хэшированием оно преобразуется в строку.

Листинг 3.79. Синтаксис функции CRYPT_HASH

```
CRYPT_HASH (<значение> USING <алгоритм>)

<алгоритм> ::= MD5 | SHA1 | SHA256 | SHA512 | SHA3_224 | SHA3_256 | SHA3_384 | SHA3_
512
```

- `Значение` - Выражение или значение любого типа. Нестроковые и небинарные типы будут преобразованы в строку. Функция поддерживает тип данных `BLOB`.
- `Алгоритм` - Криптографический алгоритм хэширования, который нужно применить.

Функция возвращает `VARBINARY`, длина которого зависит от указанного алгоритма.

Алгоритмы `MD5` и `SHA1` не рекомендуется использовать, они предоставляются только для обратной совместимости.

При хэшировании строк или двоичных значений нужно учитывать влияние пробелов и `NULL`. Хэши значения `'ab'` в `CHAR(5)` (3 пробела в конце строки), в `VARCHAR(5)` (без пробелов в конце строки)

и CHAR(6) (4 пробела в конце строки) будут отличаться. Чтобы избежать этого, нужно использовать тип данных переменной длины, или тот же тип данных фиксированной длины. Также можно нормализовать значение перед хэшированием, например, используя TRIM(TRAILING FROM <значение>).

HASH_CP

Функция возвращает хэш-значение, соответствующее входной строке, используя криптографический плагин и алгоритм хэширования, указанный в параметре конфигурации HashMethod (по умолчанию ГОСТ Р 34.11-94)

Листинг 3.80. Синтаксис функции HASH_CP

```
HASH_CP(<входной параметр>)
```

Функция поддерживает тип данных BLOB.

См. также функцию HASH.

HASHAGG

Агрегатная функция. Можно использовать в качестве оконной. Функция возвращает хэш всех элементов выборки, которые не равны NULL. При пустой выборке, или при выборке из одних NULL функция возвратит NULL.

Листинг 3.81. Синтаксис функции HASHAGG

```
HASHAGG [ALL | DISTINCT] (<выражение>) [FILTER (WHERE <условие>)]
[OVER ({<спецификация окна> | <имя окна>})]
```

Выражение может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

```
SELECT
  dept_no,
  HASHAGG(salary)
FROM employee
GROUP BY dept_no;
```

См. также агрегатные функции MIN, MAX, AVG, SUM, LIST, Оконные функции.

3.1.13 Криптографические функции

В РЕД Базе Данных 5.0 поддерживается только подмножество симметричных алгоритмов шифрования (как блочных так и потоковых) и RSA.

CRC32

Функция CRC32 возвращает CRC-32 с полиномом 0x04C11DB7. Функция может работать с любым типом данных.

Листинг 3.82. Синтаксис функции CRC32

```
CRC32(<данные>)
```

Тип возвращаемого результата: VARCHAR

ENCRYPT

Функция ENCRYPT шифрует данные с использованием симметричного шифра.

Листинг 3.83. Синтаксис функции ENCRYPT

```
ENCRYPT (<строка> [USING <алгоритм шифрования>] [MODE <режим шифрования>]
    KEY <ключ шифрования> [IV <вектор инициализации>]
    [<порядок байтов счётчика>] [CTR_LENGTH <длина счётчика>]
    [COUNTER <начальное значение счётчика>])

<алгоритм шифрования> ::= { <блочные алгоритмы> | <потоковые алгоритмы> }

<блочные алгоритмы> ::=
       { AES | ANUBIS | BLOWFISH | KHAZAD | RC5 | RC6 | SAFER+ | TWOFISH | XTEA }

<потоковые алгоритмы> ::= { CHACHA20 | RC4 | SOBER128 }

<режим шифрования> ::= { CBC | CFB | CTR | ECB | OFB }

<порядок байтов счётчика> ::= { CTR_BIG_ENDIAN | CTR_LITTLE_ENDIAN }
```

Тип возвращаемого результата: BLOB или VARBINARY

Здесь:

- <строка> — выражение строкового типа или BLOB, которое необходимо зашифровать. Размеры строк передаваемых в эту функцию должны соответствовать требованиям выбранного алгоритма и режима.
- <режим шифрования> обязателен для блочных алгоритмов шифрования.
- <вектор инициализации> должен быть указан для всех блочных алгоритмов шифрования за исключением ECB и всех потоковых алгоритмов шифрования за исключением RC4.
- <порядок байтов счётчика> может быть указан только в режиме CTR. По умолчанию используется CTR_LITTLE_ENDIAN.
- <длина счётчика> (в байтах) может быть указана только в режиме CTR. По умолчанию равна длине вектора инициализации IV.
- <начальное значение счётчика> может быть указана только для алгоритма CHACHA20. По умолчанию равно 0.

Эта функция возвращает BLOB SUB_TYPE BINARY, если первым аргументом является BLOB, и VARBINARY для всех других текстовых и двоичных типов.

Размеры строк, передаваемых в эту функцию должны соответствовать требованиям выбранного алгоритма и режима. Как правило, размер вектора инициализации должен соответствовать размеру блока алгоритма. Для режимов ECB и CBC входная строка должна быть кратной размеру блока, её нужно будет вручную заполнить нулями или пробелами, если это необходимо.

Таблица 3.8 — Требования алгоритмов шифрования

Алгоритм	Размер ключа (байт)	Размер блока (байт)	Примечание
AES	16, 24, 32	16	
ANUBIS	16 - 40 с шагом 4	16	
BLOWFISH	8 - 56	8	
KHAZAD	16	8	
RC5	8 - 128	8	
RC6	8 - 128	16	
SAFER+	16, 24, 32	16	
TWOFISH	16, 24, 32	16	
XTEA	16	8	
CHACHA20	16, 32	1	Размер (IV) составляет 8 или 12 байт. Для размера 8 initial_counter - это 64-битное целое число, для размера 12 - 32-битное.
RC4	5 - 256	1	
SOBER128	4x	1	Размер (IV) составляет 4y байт, длина не зависит от размера ключа.

Использование функции ENCRYPT:

```
select encrypt('897897' using sober128 key 'AbcdAbcdAbcdAbcd' iv '01234567')
from rdb$database;
```

DECRYPT

Функция DECRYPT дешифрует данные с использованием симметричного шифра.

Листинг 3.84. Синтаксис функции DECRYPT

```
DECRYPT (<строка> [USING <алгоритм шифрования>] [MODE <режим шифрования>]
    KEY <ключ шифрования> [IV <вектор инициализации>]
    [ <порядок байтов счётчика>] [CTR_LENGTH <длина счётчика>]
    [COUNTER <начальное значение счётчика>])

<алгоритм шифрования> ::= { <блочные алгоритмы> | <потоковые алгоритмы> }

<блочные алгоритмы> ::=
         { AES | ANUBIS | BLOWFISH | KHAZAD | RC5 | RC6 | SAFER+ | TWOFISH | XTEA }

<потоковые алгоритмы> ::= { CHACHA20 | RC4 | SOBER128 }

<режим шифрования> ::= { CBC | CFB | CTR | ECB | OFB }

<порядок байтов счётчика> ::= { CTR_BIG_ENDIAN | CTR_LITTLE_ENDIAN }
```

Тип возвращаемого результата: BLOB или VARBINARY

Здесь:

- `<строка>` — выражение строкового типа или `BLOB`, которое необходимо зашифровать. Размеры строк передаваемых в эту функцию должны соответствовать требованиям выбранного алгоритма и режима.
- `<режим шифрования>` обязателен для блочных алгоритмов шифрования.
- `<вектор инициализации>` должен быть указан для всех блочных алгоритмов шифрования за исключением `ECB` и всех потоковых алгоритмов шифрования за исключением `RC4`.
- `<порядок байтов счётчика>` может быть указан только в режиме `CTR`. По умолчанию используется `CTR_LITTLE_ENDIAN`.
- `<длина счётчика>` (в байтах) может быть указана только в режиме `CTR`. По умолчанию равна длине вектора инициализации `IV`.
- `<начальное значение счётчика>` может быть указана только для алгоритма `CHACHA20`. По умолчанию равно 0.

RSA_PRIVATE

Функция `RSA_PRIVATE` возвращает `RSA` закрытый ключ заданной длины (в байтах) в `PKCS#1` формате как строку `VARBINARY`.

Листинг 3.85. Синтаксис функции `RSA_PRIVATE`

```
RSA_PRIVATE (<размер ключа>)
```

Тип возвращаемого результата: VARBINARY

RSA_PUBLIC

Функция `RSA_PUBLIC` возвращает `RSA` открытый ключ для заданного `RSA` закрытого ключа. Оба ключа должны быть в `PKCS#1` формате.

Листинг 3.86. Синтаксис функции `RSA_PUBLIC`

```
RSA_PUBLIC (<RSA закрытый ключ>)
```

Тип возвращаемого результата: VARBINARY

RSA_ENCRYPT

Заполняет данные, используя заполнение OAEP, и шифрует их, используя открытый ключ RSA. Обычно используется для шифрования коротких симметричных ключей, которые затем используются в блочных шифрах для шифрования сообщения.

Листинг 3.87. Синтаксис функции `RSA_ENCRYPT`

```
RSA_ENCRYPT (<данные> KEY <открытый RSA ключ> [LPARAM <тег>] [HASH <алгоритм
хэширования>])

<алгоритм хэширования> ::= { MD5 | SHA1 | SHA256 | SHA512 }
```

Тип возвращаемого результата: VARBINARY

Здесь:

- <данные> — строка или BLOB для шифрования.
- <открытый RSA ключ> — открытый RSA ключ, который возвращает функция RSA_PUBLIC.
- <тег> — дополнительный системный тег, который можно применять для определения того, какая система закодировала сообщение. Значением по умолчанию является NULL.
- <алгоритм хэширования> по умолчанию SHA256.

RSA_DECRYPT

Расшифровывает с использованием закрытого ключа RSA, и удаляет OAEP дополненные данные.

Листинг 3.88. Синтаксис функции RSA_DECRYPT

```
RSA_DECRYPT (<данные> KEY <закрытый RSA ключ> [LPARAM <тег>] [HASH <алгоритм
хэширования>])

<алгоритм хэширования> ::= { MD5 | SHA1 | SHA256 | SHA512 }
```

Тип возвращаемого результата: VARCHAR

Здесь:

- <данные> — строка или BLOB для дешифрования.
- <закрытый RSA ключ> — закрытый RSA ключ, который возвращает функция RSA_PRIVATE.
- <тег> — дополнительный системный тег, который должен быть тем же самым значением, которое передавалось RSA_ENCRYPT. Если оно не совпадает с тем, который использовался во время кодирования, эта функция не расшифровывает пакет. Значением по умолчанию является NULL.
- <алгоритм хэширования> по умолчанию SHA256.

RSA_SIGN

Выполняет PSS-кодирование дайджеста сообщения для подписи и подписывает его с использованием закрытого ключа RSA. Возвращает подпись сообщения.

Листинг 3.89. Синтаксис функции RSA_SIGN

```
RSA_SIGN (<данные> KEY <закрытый RSA ключ> [HASH <алгоритм хэширования>] [SALT_LENGTH
<длина>])

<алгоритм хэширования> ::= { MD5 | SHA1 | SHA256 | SHA512 }
```

Тип возвращаемого результата: VARBINARY

Здесь:

- <данные> — строка или BLOB для кодирования.
- <закрытый RSA ключ> — закрытый RSA ключ, который возвращает функция RSA_PRIVATE.
- <алгоритм хэширования> по умолчанию SHA256.
- <длина> указывает на длину желаемой соли и, как правило, должен быть небольшим. Хорошее значение от 8 до 16.

RSA_VERIFY

Выполняет PSS-кодирование дайджеста сообщения для подписи и проверяет его цифровую подпись, используя открытый ключ RSA. Возвращает результат проверки подписи.

Листинг 3.90. Синтаксис функции RSA_SIGN

```
RSA_VERIFY (<данные>
            SIGNATURE <подпись>
            KEY <открытый RSA ключ>
            [HASH <алгоритм хэширования>]
            [SALT_LENGTH <длина>])

<алгоритм хэширования> ::= { MD5 | SHA1 | SHA256 | SHA512 }
```

Тип возвращаемого результата: BOOLEAN

Здесь:

- <данные> — строка или BLOB для кодирования.
- <подпись> должно быть значением возвращаемым функцией RSA_SIGN.
- <открытый RSA ключ> — открытый RSA ключ, который возвращает функция RSA_PUBLIC.
- <алгоритм хэширования> по умолчанию SHA256.
- <длина> указывает на длину желаемой соли и, как правило, должен быть небольшим. Хорошее значение от 8 до 16.

3.1.14 Функции преобразования типов

CAST

Функция CAST позволяет преобразовывать исходные данные из одного типа данных в другой, допустимый для исходного значения. Синтаксис функции:

Листинг 3.91. Синтаксис функции CAST

```
CAST ({<значение> | NULL} AS <тип данных>    [CHARACTER SET <набор символов>])

<тип данных> ::= {
              <тип данных SQL>
            | [TYPE OF] <имя домена>
            | TYPE OF COLUMN <имя таблицы/представления>.<имя столбца> }
```

Преобразование NULL в любой тип данных всегда дает тот же NULL.

Значением здесь может быть имя столбца таблицы, литерал или выражение.

Тип данных BLOB подтипа TEXT также допускает преобразования (но с максимальным размером 32765 байт).

В целочисленные типы данных (SMALLINT, INTEGER, BIGINT) можно выполнять преобразование числовых данных и констант с фиксированной точкой (DECIMAL, NUMERIC), с плавающей точкой (FLOAT, DOUBLE PRECISION), данных текстового BLOB и строковых данных (CHAR, VARCHAR, NCHAR и NCHAR VARYING), содержащих только цифры и десятичную точку.

В дробные числа с фиксированной точкой (DECIMAL, NUMERIC) можно преобразовывать все целочисленные данные и данные с фиксированной или плавающей точкой, данные типа BLOB подтипа TEXT, а также строки, содержащие данные, по форме соответствующие числам.

В строковые типы данных (CHAR, VARCHAR, NCHAR и NCHAR VARYING) можно преобразовывать любой тип данных. Необходимо лишь указать размер строкового типа, достаточный для того, чтобы в него поместился результат преобразования.

В типы данных DATE, TIME и TIMESTAMP можно преобразовать любую строку, содержащую дату в одном из допустимых форматов.

Более подробно типы данных, предварительно определенные литералы и контекстные переменные, а также примеры их преобразования описаны в главе 3.

См. также функцию EXTRACT.

3.1.15 Функции побитовых операций

BIN_AND

Функция BIN_AND возвращает результат побитовой операции AND над аргументами.

Листинг 3.92. Синтаксис функции BIN_AND

```
BIN_AND (<целое число>, <целое число> [, <целое число> ...])
```

Тип возвращаемого результата: SMALLINT, INTEGER, BIGINT или INT128.

Результат SMALLINT возвращается только в том случае, если все аргументы являются явными значениями типа SMALLINT или NUMERIC(n, 0), где n <= 4. В противном случае для небольших целых чисел возвращается результат типа INTEGER.

BIN_NOT

Возвращает результат логического отрицания к каждому биту двоичного представления целочисленного параметра.

Листинг 3.93. Синтаксис функции BIN_NOT

```
BIN_NOT(<значение>)
```

Выходной параметр — результат выполнения логической функции отрицания над целочисленным параметром. Если входной параметр имеет значение NULL, то возвращается также пустое значение.

Пример:

Пусть есть целое число 6, которое в двоичном представлении имеет вид 0110. Результатом операции логического отрицания будет 1001, что является дополнительным кодом для отрицательного числа -7.

```
select BIN_NOT(6) from rdb$database;
----------------------------------------
-7
```

BIN_OR

Функция BIN_OR возвращает результат побитовой операции OR (ИЛИ) над аргументами.

Листинг 3.94. Синтаксис функции BIN_OR

```
BIN_OR (<целое число>, <целое число> [, <целое число> ...])
```

Тип возвращаемого результата: SMALLINT, INTEGER, BIGINT или INT128.

Результат SMALLINT возвращается только в том случае, если все аргументы являются явными значениями типа SMALLINT или NUMERIC(n, 0), где n <= 4. В противном случае для небольших целых чисел возвращается результат типа INTEGER.

BIN_SHL

Возвращает результат операции сдвига влево двоичных знаков целочисленного параметра.

Листинг 3.95. Синтаксис функции BIN_SHL

```
BIN_SHL(<значение 1>, <значение 2>)
```

Значение входного числа сдвигается влево на количество битов, заданных вторым входным параметром. Выходной параметр — целое число, результат сдвига влево значения первого параметра на заданное количество битов, заданных вторым параметром. Если любой из входных параметров или оба имеют значение NULL, то возвращается также пустое значение.

Пример:

Пусть есть целое число 2, которое в двоичной системе имеет вид 0010. Если сделать сдвиг влево на 2 бита, то получим число 1000 = 8.

```
select BIN_SHL(2,2) from rdb$database;
----------------------------------------
8
```

Доступна для DSQL, PSQL.

BIN_SHR

Возвращает результат операции сдвига вправо двоичных знаков целочисленного параметра.

Листинг 3.96. Синтаксис функции BIN_SHR

```
BIN_SHR(<значение 1>, <значение 2>)
```

Значение входного числа сдвигается вправо на количество битов, заданных вторым входным параметром. Выходной параметр — целое число, результат сдвига вправо значения первого параметра на заданное количество битов, заданных вторым параметром. Если любой из входных параметров или оба имеют значение NULL, то возвращается также пустое значение.

Пример:

Пусть есть целое число 7, которое в двоичной системе имеет вид 0111. Если сделать сдвиг вправо на 1 бит, то получим число 0011 = 3. Поэтому

```
select BIN_SHR(7,1) from rdb$database;
----------------------------------------
3
```

Доступна для DSQL, PSQL.

BIN_XOR

Функция BIN_XOR возвращает результат побитовой операции XOR для аргументов.

Листинг 3.97. Синтаксис функции BIN_XOR

```
BIN_XOR (<целое число>, <целое число> [, <целое число> ...])
```

Тип возвращаемого результата: SMALLINT, INTEGER, BIGINT или INT128.

Результат SMALLINT возвращается только в том случае, если все аргументы являются явными значениями типа SMALLINT или NUMERIC(n, 0), где n <= 4. В противном случае для небольших целых чисел возвращается результат типа INTEGER.

3.1.16 Функции для работы с UUID

CHAR_TO_UUID

Функция для работы с UUID. Данная функция преобразует переданное в качестве параметра 32-х символьное ASCII представление UUID (XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX) в восьмеричное представление, оптимизированное для хранения.

Листинг 3.98. Синтаксис функции CHAR_TO_UUID

```
CHAR_TO_UUID(<string>)
```

Было обнаружено, что в версиях Firebird до 2.5.2 функции CHAR_TO_UUID и UUID_TO_CHAR работали неправильно на серверах с архитектурой "big-endian". В этих машинах байты/символы менялись местами и переходили в чужие позиции при преобразовании. Эта ошибка была исправлена в версиях 2.5.2 и 3.0.

```
select CHAR_TO_UUID('93519227-8D50-4E47-81AA-8F6678C096A1') from rdb$database;
----------------------------------------
935192278D504E4781AA8F6678C096A1
```

См. также функции GEN_UUID, UUID_TO_CHAR.

GEN_UUID

Функция возвращает универсальный уникальный идентификатор ID в виде 16-байтной строки символов, отвечающий требованиям стандарта RFC-4122. Функция возвращает строку UUID 4-ой версии, где несколько битов зарезервированы, а остальные являются случайными.

Синтаксис функции:

Листинг 3.99. Синтаксис функции GEN_UUID

```
GEN_UUID()
```

Тип возвращаемого результата: BINARY(16).

```
SELECT GEN_UUID() AS GUID FROM RDB$DATABASE

GUID
========
017347BFE212B2479C00FA4323B36320
```

См. также функции CHAR_TO_UUID, UUID_TO_CHAR.

UUID_TO_CHAR

Функция для работы с UUID. Данная функция преобразует переданное в качестве параметра восьмеричное представление UUID CHAR(16) в 32-х символьное ASCII представление (XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX). Тип возвращаемого значения CHAR(36).

Листинг 3.100. Синтаксис функции UUID_TO_CHAR

```
UUID_TO_CHAR(<uuid>)
```

Было обнаружено, что в версиях Firebird до 2.5.2 функции CHAR_TO_UUID и UUID_TO_CHAR работали неправильно на серверах с архитектурой "big-endian". В этих машинах байты/символы менялись местами и переходили в чужие позиции при преобразовании. Эта ошибка была исправлена в версиях 2.5.2 и 3.0.

```
select UUID_TO_CHAR(GEN_UUID()) from rdb$database;
```

См. также функции GEN_UUID, CHAR_TO_UUID.

3.1.17 Функции для работы с генераторами

GEN_ID

Функция для работы с генераторами. Позволяет получить значение, хранящееся в генераторе. Синтаксис функции:

Листинг 3.101. Синтаксис функции GEN_ID

```
GEN_ID(<имя генератора>, <приращение>)
```

При выполнении функции выбирается текущее значение указанного генератора, изменяется на величину приращения (это может быть положительное, отрицательное число или ноль). Полученное значение функция возвращает вызвавшему ее программному компоненту.

Обычно функция используется в операторах INSERT для получения уникальных числовых значений для искусственного первичного ключа.

Вместо функции GEN_ID рекомендуется использовать конструкцию NEXT VALUE FOR. Выполнение функции

```
GEN_ID (<имя генератора>, 1)
```

эквивалентно выполнению следующей конструкции:

```
NEXT VALUE FOR <имя генератора>
```

Если значение приращения меньше нуля, произойдёт уменьшение значения генератора. Следует быть крайне аккуратным при таких манипуляциях в базе данных, они могут привести к потере целостности данных.

NEXT VALUE FOR

Конструкция (функция) NEXT VALUE FOR позволяет получить значение указанного генератора, увеличенное на единицу. Синтаксис:

Листинг 3.102. Синтаксис функции NEXT VALUE FOR

```
NEXT VALUE FOR <имя генератора>
```

Точно такой же результат можно получить, вызвав функцию:

```
GEN_ID(<имя генератора>, 1)
```

Подробное описание дано в главе Генераторы.

См. также операторы CREATE GENERATOR, CREATE SEQUENCE, DROP GENERATOR, DROP SEQUENCE, SET GENERATOR, ALTER SEQUENCE, функцию GEN_ID.

3.1.18 Условные функции

CASE-WHEN-ELSE

Условное выражение. Дает возможность выбрать результирующее значение из множества различных выражений.

Листинг 3.103. Синтаксис простого выражения CASE

```
CASE <исходное выражение>
 WHEN <выражение 1> THEN {<результат 1> | NULL}
 [WHEN <выражение 2> THEN {<результат 2> | NULL}] ...
 [ELSE {<значение по умолчанию> | NULL}]
END
```

Исходное выражение возвращает значение, с которым сравниваются выражения N в последующих предложениях WHEN. Если исходное выражение равно выражению N в соответствующем предложении WHEN, то функция возвращает результат N или NULL, если пустое значение указано в этом предложении.

Если ни одно выражение N в списке предложений WHEN не равно исходному выражению, то, в случае присутствия предложения ELSE, возвращается значение по умолчанию (или NULL, если именно оно указано в этом предложении). Если же в этом случае отсутствует предложение ELSE, то функция возвращает значение NULL.

Если исходное выражение имеет значение NULL, то оно не будет соответствовать ни одному из выражений N, даже тем, которые имеют значение NULL.

Есть еще один синтаксический вариант функции CASE — поисковый CASE:

Листинг 3.104. Синтаксис поискового выражения CASE

```
CASE
 WHEN <логическое выражение> THEN {<результат> | NULL}
 [WHEN <логическое выражение> THEN {<результат> | NULL}] ...
 [ELSE {<выражение по умолчанию> | NULL}]
END
```

Здесь `<логическое выражение>` даёт тройной логический результат: `TRUE, FALSE` или `NULL`. Первое выражение, возвращающее `TRUE`, определяет результат. Если нет выражений, возвращающих `TRUE`, то в качестве результата берётся `<выражение по умолчанию>` из ветви `ELSE`. Если нет выражений, возвращающих `TRUE`, и ветвь `ELSE` отсутствует, результатом будет `NULL`.

В этих операторах `CASE`, результаты не должны быть литеральным значением: они могут быть полями или именами переменных, сложными выражениями, или иметь значение `NULL`.

См. также функции IIF, DECODE, оператор IF-THEN-ELSE.

COALESCE

Возвращает первое по порядку непустое значение в списке.

Листинг 3.105. Синтаксис функции COALESCE

```
COALESCE (<выражение 1>, <выражение 2> [, <выражение 3>]...)
```

Выполняется просмотр выражений в списке слева направо. Функция возвращает первое встретившееся непустое значение (`NOT NULL`). Если все выражения в списке имеют пустое значение, то функция возвращает `NULL`.

См. также функции CASE-WHEN-ELSE, NULLIF, IIF, операторы IF-THEN-ELSE, WHILE-DO.

DECODE

Является сокращенным вариантом функции `CASE-WHEN-ELSE`. Дает возможность выбрать возвращаемое значение из множества различных выражений.

Листинг 3.106. Синтаксис функции DECODE

```
DECODE(<исходное выражение>,
 <выражение 1>, {<результат 1> | NULL}
 [, <выражение 2>, {<результат 2> | NULL}]...
 [{<значение по умолчанию> | NULL}])
```

Исходное выражение возвращает значение, с которым сравниваются выражения N в последующих параметрах. Если исходное выражение равно выражению N в соответствующем параметре, то функция возвращает результат N или `NULL`, если пустое значение указано в этом предложении.

Если ни одно выражение N в списке не равно исходному выражению, то, в случае присутствия последнего элемента в списке, возвращается значение по умолчанию (или `NULL`, если именно оно указано в этом значении).

См. также функции IIF, CASE-WHEN-ELSE, оператор IF-THEN-ELSE.

IIF

Проверяет условие, если оно истинно, то возвращает первое значение, иначе — второе.

Листинг 3.107. Синтаксис функции IIF

```
IIF (<условие>, <значение 1> <значение 2>)
```

См. также функции CASE-WHEN-ELSE, NULLIF, COALESCE, операторы IF-THEN-ELSE, WHILE-DO.

MAXVALUE

Отыскивает максимальное значение в заданном списке. Применима к любому типу данных. Синтаксис функции:

Листинг 3.108. Синтаксис функции MAXVALUE

```
MAXVALUE (<значение> [, <значение>]...)
```

См. также функции MIN, MAX, MINVALUE, GREATEST, LEAST.

MINVALUE

Отыскивает минимальное значение в заданном списке. Применима к любому типу данных. Синтаксис функции:

Листинг 3.109. Синтаксис функции MINVALUE

```
MINVALUE (<значение> [, <значение>]...)
```

См. также функции MIN, MAX, MAXVALUE, GREATEST, LEAST.

NULLIF

Проверяет два значения на равенство, если они равны, возвращает NULL, иначе первое значение.

Листинг 3.110. Синтаксис функции NULLIF

```
NULLIF(<значение 1>, <значение 2>)
```

См. также функции CASE-WHEN-ELSE, IIF, COALESCE, операторы IF-THEN-ELSE, WHILE-DO.

GREATEST

Возвращает максимальное значение в заданном списке. Применима к любому типу данных. Синтаксис функции:

Листинг 3.111. Синтаксис функции GREATEST

```
GREATEST(<значение> [, <значение>]...)
```

См. также функции MIN, MAX, MAXVALUE, MINVALUE, LEAST.

LEAST

Возвращает минимальное значение в заданном списке. Применима к любому типу данных. Синтаксис функции:

Листинг 3.112. Синтаксис функции LEAST

```
LEAST(<значение> [, <значение>]...)
```

См. также функции MIN, MAX, MINVALUE, MAXVALUE, GREATEST.

3.1.19 Функции мониторинга ЦП

CPU_LOAD

Возвращает среднюю загрузку процессора в течение заданного интервала. Синтаксис выглядит следующим образом:

Листинг 3.113. Синтаксис функции CPU_LOAD

```
CPU_LOAD(<интервал>)
```

Функция выполняет два измерения загрузки процессора, между которыми спит в течение указанного интервала (в миллисекундах). По умолчанию тайм-аут — 1 секунда.

```
select CPU_LOAD(500) from rdb$database;
```

3.1.20 Функции для работы с LDAP

LDAP_ATTR

Читает указанный аттрибут из LDAP записи текущего пользователя.

Листинг 3.114. Синтаксис функции LDAP_ATTR

```
LDAP_ATTR(<имя атрибута>)
```

Чтобы использовать эту функцию, параметры аутентификации LDAP должны быть указаны в `firebird.conf`.

```
select LDAP_ATTR('mail') from rdb$database;
```

3.2 Агрегатные функции

Агрегатные функции выполняют вычисление на наборе значений и возвращают одиночное значение. Агрегатные функции, за исключением `COUNT`, не учитывают значения `NULL`. Агрегатные функции часто используются совместно с предложением `GROUP BY`.

Агрегатные функции могут быть использованы в качестве выражений только в следующих случаях:

- Список выбора инструкции `SELECT` (вложенный или внешний запрос);
- Предложение `HAVING`.

Обобщённый синтаксис агрегатных функций выглядит следующим образом:

Листинг 3.115. Обобщённый синтаксис агрегатных функций

```
<агрегатная функция>([ALL | DISTINCT] <выражение>) [FILTER (WHERE <условие>)] [OVER ({
<спецификация окна> | <имя окна>})]
```

3.2.1 Предложение FILTER

Предложение `FILTER` расширяет агрегатные функции дополнительным предложением `WHERE`. Если используется предложение `FILTER`, то результат агрегата строится только из строк, которые также удовлетворяют условию в дополнительном предложении `WHERE`.

Как правило, предложение фильтра может быть реализовано с использованием выражения `CASE` внутри агрегатной функции: условие фильтра должно быть помещено в предложение `WHEN`, значение, которое должно быть агрегировано в предложение `THEN`. Поскольку агрегатные функции обычно пропускают значения `NULL`, неявное предложение `ELSE NULL` достаточно, чтобы игнорировать не подходящие под условия фильтрации строки. Следующие два выражения эквивалентны:

```
SUM(<выражение>) FILTER(WHERE <условие>)
```

и

```
SUM(CASE WHEN <условие> THEN <выражение> END)
```

Для `COUNT(*)` этот пример выглядит иначе, потому что выражение `*` не может быть использовано в предложении `THEN`. Вместо этого обычно используется любое константное значение не равное `NULL`.

```
COUNT(*) FILTER(WHERE <условие>)
```

и

```
SUM(CASE WHEN <условие> THEN 1 END)
```

```
SELECT
  invoice_year,
  SUM(revenue) FILTER (WHERE invoice_month = 1) AS jan_revenue,
  SUM(revenue) FILTER (WHERE invoice_month= 2) AS feb_revenue,
  ...
  SUM(revenue) FILTER (WHERE invoice_month = 12) AS dec_revenue
FROM (
  SELECT
    EXTRACT(YEAR FROM invoices.invoice_date) AS invoice_year,
    EXTRACT(MONTH FROM invoices.invoice_date) AS invoice_month, invoices.revenue AS
revenue
  FROM invoices
)
GROUP BY invoice_year;
```

3.2.2 AVG

Агрегатная функция. Можно использовать в качестве оконной. Вычисляет среднее значение среди множества значений числового столбца или выражения. Синтаксис:

Листинг 3.116. Синтаксис функции AVG

```
AVG([ALL | DISTINCT] <выражение>) [FILTER (WHERE <условие>)]
[OVER ({<спецификация окна> | <имя окна>})]
```

Тип возвращаемого результата: тот же что и аргумент функции <выражение>

Ключевое слово ALL (принимается по умолчанию) означает, что в подсчете должны принимать участие все непустые значения, полученные оператором SELECT.

Ключевое слово DISTINCT указывает, что из исходных значений для подсчета среднего значения должны исключаться дублирующие значения.

Выражение может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

Если при выполнении оператора SELECT было получено нулевое количество записей или содержит только значения NULL, то функция возвращает пустое значение NULL.

```
SELECT
  dept_no,
  AVG(salary)
FROM employee
GROUP BY dept_no;
```

См. также агрегатные функции MIN, COUNT, SUM, LIST, Оконные функции.

3.2.3 COUNT

Агрегатная функция. Можно использовать в качестве оконной. Подсчитывает количество строк таблицы, которые удовлетворяют условию выборки данных. Синтаксис:

Листинг 3.117. Синтаксис функции COUNT

```
COUNT ({* | [ALL | DISTINCT] <выражение>}) [FILTER (WHERE <условие>)]
[OVER ({<спецификация окна> | <имя окна>})]
```

Тип возвращаемого результата: BIGINT

При указании DISTINCT из выборки устраняются дубликаты, ALL является значением по умолчанию для всех выборки значений не NULL.

Если вместо выражения <выражение> указана звёздочка (*), то будут подсчитаны все строки. Функция COUNT(*) не принимает параметры и не может использоваться с ключевым словом DISTINCT. Для функции COUNT(*) не нужен параметр <выражение>, так как по определению она не использует сведения о каких-либо конкретных столбцах. Функция COUNT(*) возвращает количество строк в указанной таблице, не отбрасывая дублированные строки. Она подсчитывает каждую строку отдельно. При этом учитываются и строки, содержащие значения NULL.

Для пустой выборки данных или если при выборке окажутся одни значения, содержащие NULL, функция возвратит значение равное 0.

Выражение может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

```
SELECT
  dept_no,
  COUNT(*) AS cnt,
  COUNT(DISTINCT full_name) AS cnt_name
FROM employee
GROUP BY dept_no;
```

См. также агрегатные функции MIN, MAX, AVG, SUM, LIST, Оконные функции.

3.2.4 LIST

Агрегатная функция. Можно использовать в качестве оконной. Объединяет в один объект типа BLOB все значения элементов выборки, которые не равны NULL. При пустой выборке функция возвратит NULL. Синтаксис функции:

Листинг 3.118. Синтаксис функции LIST

```
LIST ([ALL | DISTINCT] <выражение> [, '<разделитель>']) [FILTER (WHERE <условие>)]
[OVER ({<спецификация окна> | <имя окна>})]
```

Тип возвращаемого результата: BLOB

Тип возвращаемого значения текстовый BLOB за исключением тех случаев, когда выражением являются BLOB других подтипов.

Ключевое слово ALL (значение по умолчанию) указывает, что в список попадают все значения выборки, не содержащие NULL. Ключевое слово DISTINCT устраняет дубликаты.

В функции можно задать разделитель — произвольный символ или группу символов, заключенные в апострофы, которые в результирующем списке будут отделять одно полученное значение от другого. Если разделитель не указан, то будет использован символ запятой. Можно в качестве разделителя задать два подряд идущих апострофа. В этом случае никакой разделитель не будет использован, значения будут соединяться друг с другом.

Выражение может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает строковый тип данных или BLOB. Поля типа дата/время и числовые преобразуются к строке. Агрегатные функции в качестве выражения не допускаются.

Порядок конкатенации строк определяется порядком чтения записей из источников, который в общем случае не определён. Для придания списку необходимого порядка вы можете предварительно упорядочить источник данных, например с помощью производной таблицы.

```
-- Получение списка, порядок не определён
SELECT LIST (display_name, '; ' )
FROM GR_WORK;
-- Получение списка в алфавитном порядке
SELECT LIST (display_name, '; ' )
FROM (SELECT display_name
      FROM GR_WORK
      ORDER BY display_name);
```

См. также агрегатные функции MIN, MAX, AVG, COUNT, SUM, Оконные функции.

3.2.5 MAX

Агрегатная функция. Можно использовать в качестве оконной. Отыскивает максимальный элемент выборки, которые не равны NULL. При пустой выборке, или при выборке из одних NULL функция возвратит NULL. Применима к любому типу данных. Синтаксис функции:

Листинг 3.119. Синтаксис функции MAX

```
MAX ([ALL | DISTINCT] <выражение>) [FILTER (WHERE <условие>)]
[OVER ({<спецификация окна> | <имя окна>})]
```

Тип возвращаемого результата: DOUBLE PRECISION или масштабируемый BIGINT в зависимости от типа аргумента функции <выражение>

Здесь <выражение> может быть имя столбца, константа, переменная, выражение, неагрегатная функция или UDF. Агрегатные функции в качестве выражения не допускаются.

Если аргумент функции строка, то функция вернёт значение, которое окажется последним в сортировке при применении COLLATE.

Параметр DISTINCT не имеет смысла при использовании функцией MAX и доступен только для совместимости со стандартом.

```
SELECT
  dept_no,
  MAX(salary)
FROM employee
GROUP BY dept_no;
```

См. также агрегатные функции MIN, MINVALUE, MAXVALUE, Оконные функции.

3.2.6 MIN

Агрегатная функция. Можно использовать в качестве оконной. Отыскивает минимальный элемент выборки, которые не равны NULL. При пустой выборке, или при выборке из одних NULL функция возвратит NULL. Применима к любому типу данных. Синтаксис функции:

Листинг 3.120. Синтаксис функции MIN

```
MIN ([ALL | DISTINCT] <выражение>) [FILTER (WHERE <условие>)]
[OVER ({<спецификация окна> | <имя окна>})]
```

Тип возвращаемого результата: тот же что и аргумент функции <выражение>

Здесь <выражение> может быть имя столбца, константа, переменная, выражение, неагрегатная функция или UDF. Агрегатные функции в качестве выражения не допускаются.

Если аргумент функции строка, то функция вернёт значение, которое окажется первым в сортировке при применении COLLATE.

Параметр DISTINCT не имеет смысла при использовании функцией MIN и доступен только для совместимости со стандартом.

См. также агрегатные функции MAX, MINVALUE, MAXVALUE, Оконные функции.

3.2.7 SUM

Агрегатная функция. Можно использовать в качестве оконной. Функция возвращает сумму элементов выборки, которые не равны NULL. При пустой выборке, или при выборке из одних NULL функция возвратит NULL.

Листинг 3.121. Синтаксис функции SUM

```
SUM ([ALL | DISTINCT] <выражение>) [FILTER (WHERE <условие>)]
[OVER ({<спецификация окна> | <имя окна>})]
```

Тип возвращаемого результата: тот же что и аргумент функции <выражение>

Здесь <выражение> может быть имя столбца, константа, переменная, выражение, неагрегатная функция или UDF, имеющие числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

ALL является опцией по умолчанию. При ней обрабатываются все значения из выборки, не содержащие NULL. При указании DISTINCT из выборки устраняются дубликаты, после осуществляется подсчёт.

```
SELECT
  dept_no,
  SUM(salary)
FROM employee
GROUP BY dept_no;
```

См. также агрегатные функции MIN, MAX, AVG, COUNT, LIST, Оконные функции.

3.3 Статистические функции

Статистические функции являются агрегатными функциями. Эти функции не учитывают значения NULL. К аргументу статистической функции не применимы параметры ALL и DISTINCT.

Статистические функции часто используются совместно с предложением GROUP BY. Любую из статистических функций можно использовать в качестве оконной.

3.3.1 CORR

Функция CORR возвращает коэффициент корреляции для пары выражений, возвращающих числовые значения.

Листинг 3.122. Синтаксис функции CORR

```
CORR(<выражение1>, <выражение2>) [FILTER (WHERE <условие>)]
[OVER ({<спецификация окна> | <имя окна>})]
```

Тип возвращаемого результата: DOUBLE PRECISION

При этом аргументы функции могут содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

В статистическом смысле, корреляция - это степень связи между переменными. Связь между переменными означает, что значение одной переменной можно в определённой степени предсказать

по значению другой. Коэффициент корреляции представляет степень корреляции в виде числа в диапазоне от -1 (высокая обратная корреляция) до 1 (высокая корреляция). Значение 0 соответствует отсутствию корреляции.

Эта функция эквивалентна:

```
COVAR_POP(<выражение1>, <выражение2>)/(STDDEV_POP(<выражение2>) * STDDEV_POP(
<выражение1>))
```

В случае если выборка записей пустая или содержит только значения NULL, результат будет содержать NULL.

См. также функции Оконные функции, COVAR_POP, COVAR_SAMP, STDDEV_POP, STDDEV_SAMP, VAR_POP, VAR_SAMP.

3.3.2 COVAR_POP

Функция COVAR_POP возвращает ковариацию совокупности пар выражений с числовыми значениями.

Листинг 3.123. Синтаксис функции COVAR_POP

```
COVAR_POP(<выражение1>, <выражение2>) [FILTER (WHERE <условие>)]
[OVER ({<спецификация окна> | <имя окна>})]
```

Тип возвращаемого результата: DOUBLE PRECISION

При этом аргументы функции могут содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

Эта функция эквивалентна такой формуле:

```
(SUM(<выражение1> * <выражение2>) - SUM(<выражение1>) * SUM(<выражение2>)/COUNT(*))/
COUNT(*)
```

В случае если выборка записей пустая или содержит только значения NULL, результат будет содержать NULL.

См. также функции COUNT, SUM, Оконные функции, CORR, COVAR_SAMP, STDDEV_POP, STDDEV_SAMP, VAR_POP, VAR_SAMP.

3.3.3 COVAR_SAMP

Функция COVAR_SAMP возвращает выборочную ковариацию пары выражений с числовыми значениями.

Листинг 3.124. Синтаксис функции COVAR_SAMP

```
COVAR_SAMP(<выражение1>, <выражение2>) [FILTER (WHERE <условие>)]
[OVER ({<спецификация окна> | <имя окна>})]
```

Тип возвращаемого результата: DOUBLE PRECISION

При этом аргументы функции могут содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

Эта функция эквивалентна такой формуле:

```
(SUM(<выражение1> * <выражение2>) - SUM(<выражение1>) * SUM(<выражение2>)/COUNT(*))/
(COUNT(*)-1)
```

В случае если выборка записей пустая, содержит только 1 запись или содержит только значения NULL, результат будет содержать NULL.

См. также функции COUNT, SUM, Оконные функции, CORR, COVAR_POP, STDDEV_POP, STDDEV_SAMP, VAR_POP, VAR_SAMP.

3.3.4 STDDEV_POP

Функция STDDEV_POP возвращает среднеквадратичное отклонение для группы. Значения NULL пропускаются.

Листинг 3.125. Синтаксис функции STDDEV_POP

```
STDDEV_POP(<выражение>) [FILTER (WHERE <условие>)]
[OVER ({<спецификация окна> | <имя окна>})]
```

Тип возвращаемого результата: DOUBLE PRECISION или NUMERIC в зависимости от типа выражения

При этом аргументы функции могут содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

Эта функция эквивалентна:

```
SQRT(VAR_POP(<выражение>))
```

В случае если выборка записей пустая или содержит только значения NULL, результат будет содержать NULL.

См. также функции Оконные функции, COVAR_POP, COVAR_SAMP, CORR, STDDEV_SAMP, VAR_POP, VAR_SAMP.

3.3.5 STDDEV_SAMP

Функция STDDEV_SAMP возвращает стандартное отклонение для группы. Значения NULL пропускаются.

Листинг 3.126. Синтаксис функции STDDEV_SAMP

```
STDDEV_SAMP(<выражение>) [FILTER (WHERE <условие>)]
[OVER ({<спецификация окна> | <имя окна>})]
```

Тип возвращаемого результата: DOUBLE PRECISION или NUMERIC в зависимости от типа выражения

При этом аргументы функции могут содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

Эта функция эквивалентна:

```
SQRT(VAR_SAMP(<выражение>))
```

В случае если выборка записей пустая, содержит только 1 запись или содержит только значения NULL, результат будет содержать NULL.

См. также функции Оконные функции, COVAR_POP, COVAR_SAMP, CORR, STDDEV_POP, VAR_POP, VAR_SAMP.

3.3.6 VAR_POP

Функция VAR_POP возвращает выборочную дисперсию для группы. Значения NULL пропускаются.

Листинг 3.127. Синтаксис функции VAR_POP

```
VAR_POP(<выражение>) [FILTER (WHERE <условие>)]
[OVER ({<спецификация окна> | <имя окна>})]
```

Тип возвращаемого результата: DOUBLE PRECISION или NUMERIC в зависимости от типа выражения

При этом аргументы функции могут содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

Эта функция эквивалентна формуле:

```
(SUM(<выражение> * <выражение>) - SUM(<выражение>) * SUM(<выражение>)/COUNT(
<выражение>))/COUNT(<выражение>)
```

В случае если выборка записей пустая или содержит только значения NULL, результат будет содержать NULL.

См. также функции COUNT, SUM, Оконные функции, COVAR_POP, COVAR_SAMP, CORR, STDDEV_SAMP, STDDEV_POP, VAR_SAMP.

3.3.7 VAR_SAMP

Функция VAR_SAMP возвращает несмещённую выборочную дисперсию для группы. Значения NULL пропускаются.

Листинг 3.128. Синтаксис функции VAR_SAMP

```
VAR_SAMP(<выражение>) [FILTER (WHERE <условие>)]
[OVER ({<спецификация окна> | <имя окна>})]
```

Тип возвращаемого результата: DOUBLE PRECISION или NUMERIC в зависимости от типа выражения

При этом аргументы функции могут содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

Эта функция эквивалентна формуле:

```
(SUM(<выражение> * <выражение>) - SUM(<выражение>) * SUM(<выражение>) / COUNT(
<выражение>))/(COUNT(<выражение>)-1)
```

В случае если выборка записей пустая, содержит только 1 запись или содержит только значения NULL, результат будет содержать NULL.

См. также функции COUNT, SUM, Оконные функции, COVAR_POP, COVAR_SAMP, CORR, STDDEV_SAMP, STDDEV_POP, VAR_POP.

3.4 Функции линейной регрессии

Агрегатные функции. Функции линейной регрессии полезны для продолжения линии тренда. Линия тренда - это, как правило, закономерность, которой придерживается набор значений. Линия тренда полезна для прогнозирования будущих значений. Этот означает, что тренд будет продолжаться и в будущем. Для продолжения линии тренда необходимо знать угол наклона и точку пересечения с осью Y. Набор линейных функций включает функции для вычисления этих значений.

В синтаксисе функций, y интерпретируется в качестве переменной, зависящей от x.

Любую функции линейной регрессии из статистических функций можно использовать в качестве оконной.

3.4.1 REGR_AVGX

Функция REGR_AVGX вычисляет среднее независимой переменной линии регрессии.

Листинг 3.129. Синтаксис функции REGR_AVGX

```
REGR_AVGX(y, x) [FILTER (WHERE <условие>)]
[OVER ({<спецификация окна> | <имя окна>})]
```

Тип возвращаемого результата: DOUBLE PRECISION

Аргументы функции могут содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

Эта функция эквивалентна:

```
SUM(CASE WHEN x IS NOT NULL AND y IS NOT NULL THEN x END):REGR_COUNT(y, x)
```

См. также функции SUM, Оконные функции, REGR_AVGY, REGR_COUNT, REGR_INTERCEPT, REGR_R2, REGR_SLOPE, REGR_SXX, REGR_SXY, REGR_SYY.

3.4.2 REGR_AVGY

Функция REGR_AVGY вычисляет среднее зависимой переменной линии регрессии.

Листинг 3.130. Синтаксис функции REGR_AVGY

```
REGR_AVGY(y, x) [FILTER (WHERE <условие>)]
[OVER ({<спецификация окна> | <имя окна>})]
```

Тип возвращаемого результата: DOUBLE PRECISION

Аргументы функции могут содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

Эта функция эквивалентна:

```
SUM(CASE WHEN x IS NOT NULL AND y IS NOT NULL THEN y END):REGR_COUNT(y, x)
```

См. также функции SUM, Оконные функции, REGR_AVGX, REGR_COUNT, REGR_INTERCEPT, REGR_R2, REGR_SLOPE, REGR_SXX, REGR_SXY, REGR_SYY.

3.4.3 REGR_COUNT

Функция REGR_COUNT возвращает количество непустых пар, используемых для создания линии регрессии.

Листинг 3.131. Синтаксис функции REGR_COUNT

```
REGR_COUNT(y, x) [FILTER (WHERE <условие>)]
[OVER ({<спецификация окна> | <имя окна>})]
```

Тип возвращаемого результата: BIGINT

Аргументы функции могут содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

Эта функция эквивалентна:

```
SUM(CASE WHEN x IS NOT NULL AND y IS NOT NULL THEN 1 END)
```

См. также функции SUM, Оконные функции, REGR_AVGX, REGR_AVGY, REGR_INTERCEPT, REGR_R2, REGR_SLOPE, REGR_SXX, REGR_SXY, REGR_SYY.

3.4.4 REGR_INTERCEPT

Функция REGR_INTERCEPT вычисляет точку пересечения линии регрессии с осью Y.

Листинг 3.132. Синтаксис функции REGR_INTERCEPT

```
REGR_INTERCEPT(y, x) [FILTER (WHERE <условие>)]
[OVER ({<спецификация окна> | <имя окна>})]
```

Тип возвращаемого результата: DOUBLE PRECISION

Аргументы функции могут содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

Эта функция эквивалентна:

```
REGR_AVGY(y, x) - REGR_SLOPE(y, x) * REGR_AVGX(y, x)
```

См. также функции Оконные функции, REGR_AVGX, REGR_AVGY, REGR_COUNT, REGR_R2, REGR_SLOPE, REGR_SXX, REGR_SXY, REGR_SYY.

3.4.5 REGR_R2

Функция REGR_R2 вычисляет коэффициент детерминации, или R-квадрат, линии регрессии.

Листинг 3.133. Синтаксис функции REGR_R2

```
REGR_R2(y, x) [FILTER (WHERE <условие>)]
[OVER ({<спецификация окна> | <имя окна>})]
```

Тип возвращаемого результата: DOUBLE PRECISION

Аргументы функции могут содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

Эта функция эквивалентна:

```
POWER(CORR(y, x), 2)
```

См. также функции CORR, Оконные функции, REGR_AVGX, REGR_AVGY, REGR_COUNT, REGR_INTERCEPT, REGR_SLOPE, REGR_SXX, REGR_SXY, REGR_SYY.

3.4.6 REGR_SLOPE

Функция REGR_SLOPE вычисляет угол наклона линии регрессии.

Листинг 3.134. Синтаксис функции REGR_SLOPE

```
REGR_SLOPE(y, x) [FILTER (WHERE <условие>)]
[OVER ({<спецификация окна> | <имя окна>})]
```

Тип возвращаемого результата: DOUBLE PRECISION

Аргументы функции могут содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

Эта функция эквивалентна:

```
COVAR_POP(y, x)/VAR_POP(CASE WHEN x IS NOT NULL AND y IS NOT NULL THEN x END)
```

См. также функции COVAR_POP, VAR_POP, Оконные функции, REGR_AVGX, REGR_AVGY, REGR_COUNT, REGR_INTERCEPT, REGR_R2, REGR_SXX, REGR_SXY, REGR_SYY.

3.4.7 REGR_SXX

Функция REGR_SXX показывает диагностическую статистику, используемую для анализа регрессии.

Листинг 3.135. Синтаксис функции REGR_SXX

```
REGR_SXX(y, x) [FILTER (WHERE <условие>)]
[OVER ({<спецификация окна> | <имя окна>})]
```

Тип возвращаемого результата: DOUBLE PRECISION

Аргументы функции могут содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

Эта функция эквивалентна:

```
REGR_COUNT(y, x) * VAR_POP(<X>)

<X> ::= CASE WHEN x IS NOT NULL AND y IS NOT NULL THEN x END
```

См. также функции VAR_POP, Оконные функции, REGR_AVGX, REGR_AVGY, REGR_COUNT, REGR_INTERCEPT, REGR_R2, REGR_SLOPE, REGR_SXY, REGR_SYY.

3.4.8 REGR_SXY

Функция REGR_SXY показывает диагностическую статистику, используемую для анализа регрессии.

Листинг 3.136. Синтаксис функции REGR_SXY

```
REGR_SXY(y, x) [FILTER (WHERE <условие>)]
[OVER ({<спецификация окна> | <имя окна>})]
```

Тип возвращаемого результата: DOUBLE PRECISION

В синтаксисе функций, y интерпретируется в качестве переменной, зависящей от x.

Аргументы функции могут содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

Эта функция эквивалентна:

```
REGR_COUNT(y, x) * COVAR_POP(y, x)
```

См. также функции COVAR_POP, Оконные функции, REGR_AVGX, REGR_AVGY, REGR_COUNT, REGR_INTERCEPT, REGR_R2, REGR_SLOPE, REGR_SXX, REGR_SYY.

3.4.9 REGR_SYY

Функция REGR_SYY показывает диагностическую статистику, используемую для анализа регрессии.

Листинг 3.137. Синтаксис функции REGR_SYY

```
REGR_SYY(y, x) [FILTER (WHERE <условие>)]
[OVER ({<спецификация окна> | <имя окна>})]
```

Тип возвращаемого результата: DOUBLE PRECISION

В синтаксисе функций, y интерпретируется в качестве переменной, зависящей от x.

Аргументы функции могут содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

Эта функция эквивалентна:

```
REGR_COUNT(y, x) * VAR_POP(<Y>)

<Y> ::= CASE WHEN x IS NOT NULL AND y IS NOT NULL THEN y END
```

См. также функции VAR_POP, Оконные функции, REGR_AVGX, REGR_AVGY, REGR_COUNT, REGR_INTERCEPT, REGR_R2, REGR_SLOPE, REGR_SXX, REGR_SXY.

3.5 Оконные функции

Оконная функция выполняет вычисления над списком строк в таблице, которые как-то относятся к текущей строке. Это сравнимо с типом вычислений, которые могут быть выполнены с помощью какой-либо агрегатной функции. Но в отличие от обычных агрегатных функций, использование оконной функции не заставляет строки группироваться в одну; строки сохраняют свои отдельные значения. Другими словами, оконная функция позволяет получить доступ более чем только к текущей строке результата запроса.

Синтаксически вызов оконной функции есть указание её имени, за которым всегда следует ключевое слово `OVER()` с возможными аргументами внутри скобок. В этом и заключается её синтаксическое отличие от обычной функции или агрегатной функции. Оконные функции могут находиться только в списке `SELECT` и предложении `ORDER BY`.

Предложение `OVER` может содержать разбивку по группам ("секционирование"), сортировку и рамку окна.

Листинг 3.138. Синтаксис оконных функций

```
<оконная функция> ::=
  <агрегатная функция> OVER <имя или определение окна>
| <оконная функция> ([<выражение> [, <выражение> ...]])
  OVER <имя или определение окна>

<имя или определение окна> ::=
(<определение окна>) | <имя существующего окна>

<тип оконной функции> ::=
  <ранжирующая функция>
| <навигационная функция>

<ранжирующая функция> ::=
  RANK
| DENSE_RANK
| PERCENT_RANK
| ROW_NUMBER
| CUME_DIST
| NTILE

<навигационная функция>
  LEAD
| LAG
| FIRST_VALUE
| LAST_VALUE
| NTH_VALUE
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
<определение окна> ::=
  [<имя существующего окна>]
    [<выражение секционирования>]
    [<выражение сортировки>]
    [<рамка окна>]

<выражение секционирования> ::=
  PARTITION BY <выражение> [, <выражение> ...]

<выражение сортировки> ::=
  ORDER BY <определение сортировки> [, <определение сортировки> ...]

<определение сортировки> ::=
  <выражение> [<определение порядка>] [<порядок null>]

<определение порядка> ::=
  ASC  | ASCENDING
| DESC | DESCENDING

<порядок null> ::=
  NULLS FIRST
| NULLS LAST

<рамка окна> ::= { RANGE | ROWS } <границы рамки>

<границы рамки> ::=
  <начало рамки>
| <рамка между>

<начало рамки> ::=
  UNBOUNDED PRECEDING
| <выражение> PRECEDING
| CURRENT ROW

<рамка между> ::=
  BETWEEN { UNBOUNDED PRECEDING | <выражение> PRECEDING
          | CURRENT ROW | <выражение> FOLLOWING }
  AND { <выражение> PRECEDING | CURRENT ROW
      | <выражение> FOLLOWING | UNBOUNDED FOLLOWING }
```

`Выражение` может содержать столбец таблицы, константу, переменную, выражение, скалярную или агрегатную функцию. Оконные функции в качестве выражения не допускаются.

`Имя существующего окна` - именованное окно, определенное с помощью пункта `WINDOW` в текущем запросе.

3.5.1 Агрегатные функции

Все агрегатные функции могут быть использованы в качестве оконных функций, при добавлении предложения `OVER`.

Допустим, у нас есть таблица `EMPLOYEE` со столбцами `FULL_NAME, DEPT_NO` и `SALARY`. Нам необходимо показать для каждого сотрудника, соответствующую ему заработную плату и процент от фонда

заработной платы. Простым запросом это решается следующим образом:

```
select
   FULL_NAME,
   DEPT_NO,
   SALARY,
   SALARY / (select sum(SALARY) from employee) percentage
from employee
order by FULL_NAME;
```

Запрос повторяется и может работать довольно долго, особенно если EMPLOYEE является сложным представлением.

Этот запрос может быть переписан в более быстрой и элегантной форме с использованием оконных функций:

```
select
   FULL_NAME,
   DEPT_NO,
   SALARY,
   SALARY / sum(SALARY) OVER () percentage
from employee
order by FULL_NAME;
```

Здесь sum(salary) OVER () вычисляет сумму всех зарплат из запроса (таблицы сотрудников).

3.5.2 Секционирование

Как и агрегатные функции, которые могут работать отдельно или по отношению к группе, оконные функции тоже могут работать для групп, которые называются "секциями" (partition) или разделами.

```
<оконная функция>(...) OVER (PARTITION BY <выражение> [, <выражение> ...])
```

Для каждой строки, оконная функция обрабатывает только строки, которые попадают в ту же самую секцию, что и текущая строка.

Агрегирование над группой может давать более одной строки, таким образом, к результирующему набору, созданному секционированием, присоединяются результаты из основного запроса, используя тот же список выражений, что и для секции.

Продолжая пример с сотрудниками, вместо того чтобы считать процент зарплаты каждого сотрудника от суммарной зарплаты сотрудников, посчитаем процент от суммарной зарплаты сотрудников того же отдела:

```
select
   FULL_NAME,
   DEPT_NO,
   SALARY,
   SALARY / sum(SALARY) OVER (PARTITION BY DEPT_NO) percentage
from employee
order by FULL_NAME;
```

3.5.3 Сортировка

Предложение ORDER BY может использоваться как с секционированием, так и без него. Предложение ORDER BY внутри OVER определяет порядок, в котором оконная функция будет обрабатывать строки. Этот порядок не обязательно должен совпадать с порядком вывода строк.

С оконными функциями связана важная особенность: для каждой строки есть набор строк в ее разделе, называемый рамкой окна. По умолчанию, при указании ORDER BY, рамка состоит из всех строк от начала раздела до текущей строки и строк, равных текущему выражению ORDER BY. Без ORDER BY рамка по умолчанию состоит из всех строк в разделе.

В результате для стандартных агрегатных функций предложение ORDER BY дает результаты агрегации частично по мере обработки строк.

```
select
   EMP_NO,
   SALARY,
   SUM(SALARY) OVER (ORDER BY SALARY) AS cumul_salary
from employee
order by SALARY;

EMP_NO SALARY CUMUL_SALARY
====== ====== =============
3      8.00   8.00
4      9.00   17.00
1      10.00  37.00
5      10.00  37.00
2      12.00  49.00
```

В этом случае cumul_salary возвращает частичную/накопительную агрегацию (функции SUM). Может показаться странным, что значение 37,00 повторяется для идентификаторов 1 и 5, но так и должно быть. Ключи ORDER BY группируются вместе, и агрегация вычисляется один раз (но суммирует два 10.00). Чтобы избежать этого, можно добавить поле ID в конец предложения ORDER BY.

Можно использовать несколько окон с разной сортировкой, а также такие пункты ORDER BY, как ASC/DESC и NULLS {FIRST | LAST}.

С секциями предложение ORDER BY работает таким же образом, но на границе каждой секции агрегаты сбрасываются.

Все агрегатные функции могут использовать ORDER BY, кроме LIST().

Следующий пример показывает сумму кредита, накопленную сумму выплат и остаток по выплатам.

```
select
   payments.id AS id,
   payments.bydate AS bydate,
   credit.amount AS credit_amount,
   payments.amount AS pay,
   SUM(payments.amount) OVER(ORDER BY payments.bydate) AS s_amount,
   SUM(payments.amount) OVER(ORDER BY payments.bydate, payments.id) AS s_amount2,
   credit.amount - SUM(payments.amount) OVER(ORDER BY payments.bydate, payments.id) AS
balance
from credit
JOIN payments ON payments.credit_id = credit.id
WHERE credit.id = 1
ORDER BY payments.bydate;
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
ID BYDATE      CREDIT_AMOUNT PAY    S_AMOUNT S_AMOUNT2 BALANCE
== =========== ============= ====== ======== ========= ===========
1 2023-01-01   1000.00       100.00 100.00   100.00    900.00
2 2023-01-05   1000.00       200.00 300.00   300.00    700.00
3 2023-01-10   1000.00       150.00 450.00   450.00    550.00
4 2023-01-15   1000.00       250.00 700.00   700.00    300.00
5 2023-01-20   1000.00       300.00 1000.00  1000.00   0.00
6 2023-01-25   1000.00        50.00 1050.00  1050.00   -50.00
```

3.5.4 Рамка окна

Набор строк внутри секции, которым оперирует оконная функция, называется рамкой окна (кадры окна). Рамка окна определяет, какие строки следует учитывать для текущей строки при оценке оконной функции

Рамка окна состоит из трёх частей: единица (`unit`), начальная граница и конечная граница. В качестве единицы могут быть использованы ключевые слова `RANGE` или `ROWS`, которые указывают, каким образом будут работать границы окна. Границы окна определяются следующими выражениями:

- `UNBOUNDED PRECEDING`
- `<выражение> PRECEDING`
- `UNBOUNDED FOLLOWING`
- `<выражение> FOLLOWING`
- `CURRENT ROW`
- При использовании `RANGE` в `ORDER BY` должно быть указано только одно выражение, и оно должно быть числового типа, `DATE`, `TIME` или `TIMESTAMP`. Для `PRECEDING` выражение вычитается из выражения в `ORDER BY`, а для `FOLLOWING` добавляется. Для `CURRENT ROW` выражение в `ORDER BY` используется как есть.

Все строки внутри секции, находящиеся между границами, считаются частью результирующей рамки окна.

- При использовании `ROWS` на предложение `ORDER BY` не накладывается ограничений на количество и типы выражений. В этом случае `<выражение> PRECEDING` указывает количество строк предшествующее текущей строке, а `<выражение> FOLLOWING` количество строк после текущей строки.

`UNBOUNDED PRECEDING` и `UNBOUNDED FOLLOWING` работают одинаково для предложений `ROWS` и `RANGE`.

Фраза `UNBOUNDED PRECEDING` указывает, что окно начинается с первой строки секции. `UNBOUNDED PRECEDING` может быть указано только как начальная точка окна.

Фраза `UNBOUNDED FOLLOWING` указывает, что окно заканчивается последней строкой секции. `UNBOUNDED FOLLOWING` может быть указано только как конечная точка окна.

Фраза `CURRENT ROW` указывает, что окно начинается или заканчивается на текущей строке при использовании совместно с предложением `ROWS` или что окно заканчивается на текущем значении при использовании с предложением `RANGE`. `CURRENT ROW` может быть задана и как начальная, и как конечная точка.

Предложение `BETWEEN` используется совместно с ключевым словом `ROWS` или `RANGE` для указания нижней (начальной) или верхней (конечной) граничной точки окна. Верхняя граница не может быть меньше нижней границы.

> Если указана только начальная точка окна, то конечной точкой окна считается `CURRENT ROW`. Например, если указано `ROWS 1 PRECEDING`, то это аналогично указанию `ROWS BETWEEN 1 PRECEDING AND CURRENT ROW`.

Некоторые оконные функции игнорируют выражение рамки:

- `ROW_NUMBER`, `LAG` и `LEAD` всегда работают как `ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW`.
- `DENSE_RANK`, `RANK`, `PERCENT_RANK` и `CUME_DIST` работают как `RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW`.
- `FIRST_VALUE`, `LAST_VALUE` и `NTH_VALUE` работают на рамке, но `RANGE` работает идентично `ROWS`.

Таким образом, предложения `ROWS` и `RANGE` позволяют довольно гибко настроить размер плавающего окна. Чаще всего встречаются следующие варианты:

- Нижняя граница фиксирована (совпадает с первой строкой упорядоченной группы), а верхняя граница ползёт (совпадает с текущей строкой упорядоченной группы). В этом случае получаем нарастающий итог (кумулятивный агрегат). В этом случае размер окна меняется (расширяется в одну сторону) и само окно движется за счёт расширения. Возможна и обратная ситуация, когда нижняя граница ползёт, а верхняя зафиксирована. В этом случае окно будет сужаться.
- Если верхняя и нижняя границы фиксированы относительно текущей строки, например 1 строка до текущей и 2 после текущей, то получаем скользящий агрегат. В этом случае размер окна фиксирован, а само окно скользит.

Окна диапазона

Окна диапазона объединяют строки в соответствии с заданным порядком. Например, если рамка окна задана выражением `RANGE 5 PRECEDING`, то будет сгенерировано перемещающееся окно, включающее предыдущие строки группы, значение которых меньше текущего не более чем на 5.

```
SELECT FIRST 5
   emp_no,
   salary,
   SUM(salary) OVER() AS s1,
   SUM(salary) OVER(ORDER BY salary) AS s2,
   SUM(salary) OVER(ORDER BY salary RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)
AS s3,
   SUM(salary) OVER(ORDER BY salary RANGE BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING)
AS s4,
   SUM(salary) OVER(ORDER BY salary RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED
FOLLOWING) AS s5,
   SUM(salary) OVER(ORDER BY salary RANGE BETWEEN CURRENT ROW AND 1 FOLLOWING) AS s6,
   SUM(salary) OVER(ORDER BY salary RANGE BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS s7,
   SUM(salary) OVER(ORDER BY salary RANGE 1 PRECEDING) AS s8
FROM employee;

EMP_NO SALARY  S1      S2      S3      S4      S5      S6      S7      S8
====== ======= ======= ======= ======= ======= ======= ======= ======= =======
3      8.00    49.00   8.00    8.00    49.00   49.00   17.00   17.00   8.00
4      9.00    49.00   17.00   17.00   41.00   49.00   29.00   37.00   17.00
1      10.00   49.00   37.00   37.00   32.00   49.00   20.00   29.00   29.00
5      10.00   49.00   37.00   37.00   32.00   49.00   20.00   29.00   29.00
2      12.00   49.00   49.00   49.00   12.00   49.00   12.00   12.00   12.00
```

Для того чтобы понять, какие значения будут входить в диапазон, можно использовать функции `FIRST_VALUE` и `LAST_VALUE`. Это помогает увидеть диапазоны окна и проверить, корректно ли установлены параметры.

Окна строк

Окна срок задаются в физических единицах, строках. Например, если рамка окна задана выражением ROWS 5 PRECEDING, то окно будет включать в себя до 6 строк: текущую и пять предыдущих (порядок определяется конструкцией ORDER BY).

```
SELECT FIRST 5
   emp_no,
   salary,
   SUM(salary) OVER() AS s1,
   SUM(salary) OVER(ORDER BY salary) AS s2,
   SUM(salary) OVER(ORDER BY salary ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)
AS s3,
   SUM(salary) OVER(ORDER BY salary ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING)
AS s4,
   SUM(salary) OVER(ORDER BY salary ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED
FOLLOWING) AS s5,
   SUM(salary) OVER(ORDER BY salary ROWS BETWEEN CURRENT ROW AND 1 FOLLOWING) AS s6,
   SUM(salary) OVER(ORDER BY salary ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS s7,
   SUM(salary) OVER(ORDER BY salary ROWS 1 PRECEDING) AS s8
FROM employee;

EMP_NO SALARY  S1      S2      S3      S4      S5      S6      S7      S8
====== ======= ======= ======= ======= ======= ======= ======= ======= =======
3      8.00    49.00   8.00    8.00    49.00   49.00   17.00   17.00   8.00
4      9.00    49.00   17.00   17.00   41.00   49.00   19.00   27.00   17.00
5      10.00   49.00   37.00   37.00   22.00   49.00   20.00   29.00   19.00
1      10.00   49.00   37.00   27.00   32.00   49.00   22.00   32.00   20.00
2      12.00   49.00   49.00   49.00   12.00   49.00   12.00   22.00   22.00
```

3.5.5 Именованные окна

Предложение WINDOW можно использовать для явного наименования окна, например, чтобы избежать повторяющихся или сложных выражений. Именованное окно можно использовать в предложении OVER для ссылки на определение окна, например OVER <имя окна>, и в качестве базы другого именованного или встроенного окна (в предложении OVER). Окна с рамкой (с предложениями RANGE и ROWS) не могут быть использованы в качестве базового окна (но могут быть использованы в предложении OVER <имя окна>).

Окно, которое ссылается на базовое окно, не может содержать предложения PARTITION BY или переопределять сортировку (ORDER BY) базового окна.

```
SELECT first 5
   emp_no,
   dept_no,
   salary,
   count(*) OVER w1,
   first_value(salary) OVER w2,
   last_value(salary) OVER w2,
   sum(salary) over (w2 ROWS BETWEEN CURRENT ROW AND 1 FOLLOWING) AS s
FROM employee
WINDOW w1 AS (PARTITION BY dept_no),
       w2 AS (w1 ORDER BY salary)
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
ORDER BY dept_no, salary;
```

3.5.6 Ранжирующие функции

Ранжирующие функции вычисляют порядковый номер ранга внутри секции окна.

Эти функции могут применяться с использованием секционирования и сортировки и без них. Однако их использование без сортировки почти никогда не имеет смысла.

Функции ранжирования могут быть использованы для создания различных типов инкрементных счётчиков. Рассмотрим `SUM(1) OVER (ORDER BY SALARY)` в качестве примера того, что они могут делать, каждая из них различным образом. Ниже приведён пример запроса, который позволяет сравнить их поведение по сравнению с `SUM`.

```
SELECT first 5
   emp_no,
   salary,
   DENSE_RANK() OVER (ORDER BY salary),
   RANK() OVER (ORDER BY salary),
   PERCENT_RANK() OVER(ORDER BY salary),
   CUME_DIST() OVER(ORDER BY salary),
   NTILE(3) OVER(ORDER BY salary),
   ROW_NUMBER() OVER (ORDER BY salary) ROW_NUM,
   SUM(1) OVER (ORDER BY salary)
FROM employee
ORDER BY salary;

EMP_NO SALARY DENSE_RANK RANK PERCENT_RANK       CUME_DIST           NTILE ROW_NUM SUM
====== ====== ========== ==== ================== ================== ===== ======= ===
3      8.00   1          1    0.000000000000000 0.2000000000000000  1     1       1
4      9.00   2          2    0.2500000000000000 0.4000000000000000 1     2       2
1      10.00  3          3    0.5000000000000000 0.8000000000000000 2     3       4
5      10.00  3          3    0.5000000000000000 0.8000000000000000 2     4       4
2      12.00  4          5    1.000000000000000 1.000000000000000   3     5       5
```

Разница между `DENSE_RANK` и `RANK` заключается в том, что только в `RANK` есть пропуски, связанные с дублированием строк (относительно упорядочивания окна). `DENSE_RANK` продолжает присваивать порядковые номера после дублирующейся зарплаты. С другой стороны, `ROW_NUMBER` всегда присваивает порядковые номера, даже при наличии дублирующихся значений.

DENSE_RANK

```
DENSE_RANK() OVER { <определение окна> | <имя окна>}
```

Тип возвращаемого результата: BIGINT

Возвращает ранг строк в секции результирующего набора без промежутков в ранжировании. Строки с одинаковыми значениями `<выражение сортировки>` получают одинаковый ранг в пределах группы `<выражение секционирования>`, если она указана. Ранг строки равен количеству различных значений рангов в секции, предшествующих текущей строке, увеличенному на единицу.

```
SELECT first 5
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
   emp_no,
   salary,
   DENSE_RANK() OVER (ORDER BY salary)
FROM employee
ORDER BY salary;

EMP_NO SALARY DENSE_RANK
====== ====== ==========
3      8.00   1
4      9.00   2
1      10.00  3
5      10.00  3
2      12.00  4
```

RANK

```
RANK() OVER {<определение окна> | <имя окна>}
```

Тип возвращаемого результата: BIGINT

Возвращает ранг каждой строки в секции результирующего набора. Строки с одинаковыми значениями `<выражение сортировки>` получают одинаковый ранг в пределах группы `<выражение секционирования>`, если она указана. Ранг строки вычисляется как единица плюс количество рангов, находящихся до этой строки.

```
SELECT FIRST 5
   emp_no,
   salary,
   RANK() OVER (ORDER BY salary)
FROM employee
ORDER BY salary;

EMP_NO SALARY RANK
====== ====== ====
3      8.00   1
4      9.00   2
1      10.00  3
5      10.00  3
2      12.00  5
```

PERCENT_RANK

```
PERCENT_RANK() OVER {<определение окна> | <имя окна>}
```

Тип возвращаемого результата: DOUBLE PRECISION

Возвращает относительный ранг текущей строки в группе строк. Функция `PERCENT_RANK` используется для вычисления относительного положения значения в секции или результирующем наборе запроса. Диапазон значений, возвращаемый функцией `PERCENT_RANK`, больше 0 и меньше или равен 1.

В первой строке любого набора PERCENT_RANK равна 0. Значения NULL по умолчанию включаются и рассматриваются как наименьшие возможные значения.

Функция PERCENT_RANK вычисляется как

```
PERCENT_RANK = (RANK-1 текущей строки)/(<число строк в разделе> - 1)
```

Другими словами, P PERCENT_RANK() OVER <имя_окна> эквивалентно (RANK() OVER <имя_окна> - 1)/ CAST(COUNT(*) OVER() - 1 AS DOUBLE PRECISION) <имя_окна> - 1) / CAST(COUNT(*) OVER() - 1 AS DOUBLE PRECISION).

```
SELECT FIRST 5
   emp_no,
   salary,
   PERCENT_RANK() OVER (ORDER BY salary)
FROM employee
ORDER BY salary;

EMP_NO SALARY PERCENT_RANK
====== ====== ==================
3      8.00   0.000000000000000
4      9.00   0.2500000000000000
1      10.00  0.5000000000000000
5      10.00  0.5000000000000000
2      12.00  1.000000000000000
```

CUME_DIST

```
CUME_DIST() OVER {<определение окна> | <имя окна>}
```

Тип возвращаемого результата: DOUBLE PRECISION

Функция CUME_DIST рассчитывает кумулятивное распределение значения внутри раздела окна. Возвращаемое значение находится в диапазоне от 0 до 1. Функция CUME_DIST рассчитывается как (число строк, предшествующих или равных текущей) / (общее число строк в разделе). Другими словами, CUME_DIST() OVER <имя_окна> эквивалентно COUNT(*) OVER <имя_окна> / COUNT(*) OVER(). Для равных значений всегда вычисляется одно и то же значение накопительного распределения. Значения NULL по умолчанию включаются и рассматриваются как наименьшие возможные значения.

```
SELECT FIRST 5
   emp_no,
   salary,
   CUME_DIST() OVER (ORDER BY salary)
FROM employee
ORDER BY salary;

EMP_NO SALARY CUME_DIST
====== ====== ==================
3      8.00   0.2000000000000000
4      9.00   0.4000000000000000
1      10.00  0.8000000000000000
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
5       10.00  0.8000000000000000
2       12.00  1.000000000000000
```

NTILE

Функция NTILE распределяет строки раздела в заданное количество групп так, чтобы размеры групп были максимально близки. Группы нумеруются, начиная с единицы. Для каждой строки функция NTILE возвращает номер группы, к которой принадлежит строка.

```
NTILE(<выражение>) OVER {<определение окна> | <имя окна>}
```

Тип возвращаемого результата: BIGINT

Выражение целочисленного типа. Указывает количество групп, на которые необходимо разделить каждую секцию. Ограничивается целым положительным литералом, именованным параметром (PSQL) или позиционным параметром (DSQL).

Если количество строк в секции не делится на <число>, то формируются группы двух размеров, отличающихся на единицу. Группы большего размера следуют перед группами меньшего размера в порядке, заданном в предложении OVER.

```
SELECT FIRST 5
   emp_no,
   salary,
   NTILE(3) OVER (ORDER BY salary)
FROM employee
ORDER BY salary;

EMP_NO SALARY NTILE
====== ====== =====
3      8.00   1
4      9.00   1
1      10.00  2
5      10.00  2
2      12.00  3
```

ROW_NUMBER

```
ROW_NUMBER() OVER {<определение окна> | <имя окна>}
```

Тип возвращаемого результата: BIGINT

Возвращает порядковый номер строки в разделе, где 1 - первая строка в каждом из разделов.

```
SELECT FIRST 5
   emp_no,
   salary,
   ROW_NUMBER() OVER (ORDER BY salary)
FROM employee
ORDER BY salary;
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
EMP_NO SALARY ROW_NUMBER
====== ====== ==========
3      8.00   1
4      9.00   2
1      10.00  3
5      10.00  4
2      12.00  5
```

3.5.7 Навигационные функции

Навигационные функции получают простые (не агрегированные) значения выражения из другой строки запроса в той же секции.

Функции `FIRST_VALUE, LAST_VALUE` и `NTH_VALUE` оперируют на рамке окна (кадрах окна). По умолчанию, если задано предложение `ORDER BY`, то рамка состоит из всех строк, от начала разбиения до текущей строки, плюс любые следующие строки, которые равны текущей строке в соответствии с предложением `ORDER BY`, т.е.

```
RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
```

Из-за этого результаты функций `NTH_VALUE` и в особенности `LAST_VALUE` могут показаться странными. Для устранения этого "недостатка" вы можете задать другую рамку окна, например:

```
ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING
```

```
SELECT FIRST 5
   emp_no,
   salary,
   FIRST_VALUE(salary) OVER (ORDER BY salary),
   LAST_VALUE(salary) OVER (ORDER BY salary),
   NTH_VALUE(salary, 2) OVER (ORDER BY salary),
   LAG(salary) OVER (ORDER BY salary),
   LEAD(salary) OVER (ORDER BY salary)
FROM employee
ORDER BY salary;

EMP_NO SALARY FIRST_VALUE LAST_VALUE NTH_VALUE  LAG    LEAD
====== ====== =========== ========== ========== ====== ======
3      8.00   8.00        8.00       <null>     <null> 9.00
4      9.00   8.00        9.00       9.00       8.00   10.00
1      10.00  8.00        10.00      9.00       9.00   10.00
5      10.00  8.00        10.00      9.00       10.00  12.00
2      12.00  8.00        12.00      9.00       10.00  <null>
```

Вариант с изменённой рамкой окна для функций `LAST_VALUE` и `NTH_VALUE`.

```
SELECT FIRST 5
   emp_no,
   salary,
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
    FIRST_VALUE(salary) OVER (ORDER BY salary),
    LAST_VALUE(salary) OVER w,
    NTH_VALUE(salary, 2) OVER w,
    LAG(salary) OVER (ORDER BY salary),
    LEAD(salary) OVER (ORDER BY salary)
FROM employee
WINDOW w AS (ORDER BY salary ROWS BETWEEN UNBOUNDED PRECEDING AND
UNBOUNDED FOLLOWING)
ORDER BY salary;

EMP_NO SALARY FIRST_VALUE LAST_VALUE NTH_VALUE  LAG    LEAD
====== ====== =========== ========== ========== ====== ======
3      8.00   8.00        12.00      9.00       <null> 9.00
4      9.00   8.00        12.00      9.00       8.00   10.00
1      10.00  8.00        12.00      9.00       9.00   10.00
5      10.00  8.00        12.00      9.00       10.00  12.00
2      12.00  8.00        12.00      9.00       10.00  <null>
```

FIRST_VALUE

```
FIRST_VALUE(<выражение>) OVER {<определение окна> | <имя окна>}
```

Тип возвращаемого результата: тот же что и аргумент функции <выражение>

Выражение может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF. Агрегатные функции в качестве выражения не допускаются.

Функция FIRST_VALUE(<выражение>) возвращает первое значение из упорядоченного набора значений.

LAG

```
LAG(<выражение> [, <смещение> [, <default>]]) OVER {<определение окна> | <имя окна>}
```

Тип возвращаемого результата: тот же что и аргумент функции <выражение>

Функция LAG обеспечивает доступ к строке с заданным физическим смещением перед началом текущей строки. Если смещение указывает за пределы секции, то будет возвращено значение <default>, которое по умолчанию равно NULL.

Параметр <выражение> может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF. Агрегатные функции в качестве выражения не допускаются.

Параметр <смещение> — количество строк до строки перед текущей строкой, из которой необходимо получить значение. Если значение аргумента не указано, то по умолчанию принимается 1. <смещение> может быть столбцом, вложенным запросом или другим выражением, с помощью которого вычисляется целая положительная величина, или другим типом, который может быть неявно преобразован в BIGINT. <смещение> не может быть отрицательным значением или аналитической функцией.

Предположим у вас есть таблица rate, которая хранит курс валюты на каждый день. Необходимо проследить динамику изменения курса за последние пять дней.

```
SELECT
   bydate,
   cost,
   cost - LAG(cost) OVER(ORDER BY bydate) AS change,
   100 * (cost - LAG(cost) OVER(ORDER BY bydate)) / LAG(cost) OVER(ORDER BY bydate) AS
percent_change
FROM rate
ORDER BY bydate;

BYDATE      COST   CHANGE PERCENT_CHANGE
=========== ====== ====== ==============
2014-10-27  31.00  <null> <null>
2014-10-28  31.53  0.53   1.7096
2014-10-29  31.40  -0.13  -0.4123
2014-10-30  31.67  0.27   0.8598
2014-10-31  32.00  0.33   1.0419
```

LAST_VALUE

```
LAST_VALUE(<выражение>) OVER {<определение окна> | <имя окна>}
```

Тип возвращаемого результата: тот же что и аргумент функции <выражение>

Функция LAST_VALUE возвращает последнее значение из упорядоченного набора значений рамки окна. Параметр <выражение> может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF. Агрегатные функции в качестве выражения не допускаются.

LEAD

```
LEAD(<выражение> [, <смещение> [, <default>]]) OVER {<определение окна> | <имя окна>}
```

Тип возвращаемого результата: тот же что и аргумент функции <выражение>

Функция LEAD обеспечивает доступ к строке на заданном физическом смещении после текущей строки. Если смещение указывает за пределы секции, то будет возвращено значение <default>, которое по умолчанию равно NULL.

Параметр <выражение> может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF. Агрегатные функции в качестве выражения не допускаются.

Параметр <смещение> — количество строк после текущей строки до строки, из которой необходимо получить значение. Если значение аргумента не указано, то по умолчанию принимается 1. <смещение> может быть столбцом, вложенным запросом или другим выражением, с помощью которого вычисляется целая положительная величина, или другим типом, который может быть неявно преобразован в BIGINT. <смещение> не может быть отрицательным значением или аналитической функцией.

NTH_VALUE

```
NTH_VALUE(<выражение> [, <смещение> ]) [FROM FIRST | FROM LAST]
OVER {<определение окна> | <имя окна>}
```

Тип возвращаемого результата: тот же что и аргумент функции <выражение>

Функция NTH_VALUE возвращает N-ое значение, начиная с первой (опция FROM FIRST) или последней (опция FROM LAST) записи. По умолчанию используется опция FROM FIRST. Смещение 1 от первой записи будет эквивалентно функции FIRST_VALUE, смещение 1 от последней записи будет эквивалентно функции LAST_VALUE.

Параметр <выражение> может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF. Агрегатные функции в качестве выражения не допускаются

Параметр <смещение> — номер записи, начиная с первой (опция FROM FIRST) или последней (опция FROM LAST) записи.

3.5.8 Агрегатные функции внутри оконных

В качестве аргументов оконных функций, а также в предложении OVER разрешено использование агрегатных функций (но не оконных). В этом случае сначала вычисляются агрегатные функции, а только затем на них накладываются окна оконных функций.

При использовании агрегатных функции в качестве аргументов оконных функций, все столбцы, не используемые в агрегатных функциях должны быть указаны в предложении GROUP BY.

Пример использования агрегатной функции в качестве аргумента оконной:

```
SELECT first 5
   emp_no,
   AVG(salary) AS avg_salary,
   RANK() OVER(ORDER BY AVG(salary)) AS salary_rank
FROM employee
GROUP BY emp_no;
```

3.6 Табличные функции

Табличные функции возвращают набор строк, содержащий поля базовых типов. Они могут использоваться в предложении FROM в качестве источника, из которого осуществляется выборка данных. Столбцы, возвращаемые табличными функциями, можно указывать в выражениях SELECT, JOIN и WHERE.

3.6.1 UNLIST

Функция UNLIST преобразует входную строку в набор записей, состоящий из одного столбца.

Синтаксис функции:

Листинг 3.139. Синтаксис функции UNLIST

```
UNLIST(<входная строка> [, <разделитель>] [<возвращаемый тип данных>]) AS <псевдоним
набора записей> [(<псевдоним результирующего столбца>)]

<разделитель> ::= <строковый литерал>

<возвращаемый тип данных> ::= RETURNING <тип данных>
```

Описание параметров функции:

- Входная строка — Входное значение, представляющее собой любое выражение, возвращающее строку символов (строковый литерал, столбец таблицы, константа, переменная, выражение и т.д). Также может быть значением BLOB TEXT. Обязательный параметр.

- `Разделитель` — Строковый литерал, заключенный в апострофы, который в результирующем списке будет отделять одно полученное значение от другого. Также может содержать выражение, возвращающее строку символов. Может быть значением `BLOB TEXT`. Если в качестве разделителя указана пустая строка, то значения будут выведены одной записью. Если разделитель не задан, то будет использован символ запятой.
- `Возвращаемый тип данных` — Тип данных, к которому будут приведены значения в результирующем наборе записей. Если возвращаемый тип не задан, то по умолчанию используется тип `VARCHAR(32)`. В качестве возвращаемого типа может быть указан домен.
- `Псевдоним набора записей` — Псевдоним набора записей, возвращаемого функцией `UNLIST`. Обязательный параметр.
- `Псевдоним результирующего столбца` — Псевдоним столбца, возвращаемого функцией `UNLIST`. Если псевдоним не указан, то по умолчанию используется `UNLIST`.

Пример работы функции:

```
SELECT * FROM UNLIST('1,2,3,4') AS A;

UNLIST
=======
1
2
3
4
```

Пример использования `UNLIST` в качестве источника данных:

```
CREATE TABLE TEST_TABLE (ID INT);
INSERT INTO TEST_TABLE (ID) SELECT * FROM UNLIST('1,2,3,4' RETURNING INT) AS A;
SELECT * FROM TEST_TABLE;

ID
=======
1
2
3
4
```

Приложение И Контекстные переменные

В SQL РЕД База Данных существует четыре предварительно определенных литерала и множество контекстных переменных. Фактически все они неявно являются функциями. Обращение к предварительно определенному литералу или к контекстной переменной является обращением к соответствующей функции, определенной в системе управления базами данных, каждая из которых возвращает одно требуемое значение.

CURRENT_CONNECTION

Контекстная переменная `CURRENT_CONNECTION` имеет тип данных `INTEGER`. Она возвращает число — системный идентификатор текущего соединения с базой данных. Значение переменной хранится в странице заголовка базы и сбрасывается после восстановления базы. Переменная увеличивается на единицу при каждом последующем соединении с базой данных (соединения также могут быть внутренними вызванными самим ядром). Следовательно, переменная показывает количество подключений произошедших к базе после её восстановления (или после её создания).

Для обычных программ работы с базой данных эта переменная вряд ли может быть полезной.

CURRENT_DATE

`CURRENT_DATE` типа `DATE` возвращает текущую дату сервера.

CURRENT_ROLE

Контекстная переменная `CURRENT_ROLE` типа `VARCHAR(63)` возвращает имя роли, под которой с базой данных соединился пользователь в операторе `CONNECT`. Если при соединении с базой данных роль не была указана, то возвращается пустое значение `NONE`.

CURRENT_TIME

`CURRENT_TIME` типа `TIME WITH TIME ZONE` возвращает текущее время сервера. Время вычисляется в момент начала выполнения запроса и не меняется до его завершения. Эта контекстная переменная возвращает не только время, но и тысячные доли секунды. В обращении к контекстной переменной `CURRENT_TIME` можно указывать и количество знаков в требуемых долях секунды:

```
CURRENT_TIME [(<количество знаков в долях секунды>)]
```

Количество знаков может быть числом от 0 до 3. Если количество знаков не указано, предполагается 0.

CURRENT_TIMESTAMP

Контекстная переменная `CURRENT_TIMESTAMP` типа `TIMESTAMP WITH TIME ZONE` возвращает текущую дату и текущее время сервера. Время вычисляется в момент начала выполнения запроса и не меняется до его завершения. В текущем времени указываются и миллисекунды - три знака после десятичной точки. При обращении к этой контекстной переменной также можно задавать требуемое количество долей секунды:

```
CURRENT_TIMESTAMP [(<количество знаков в долях секунды>)]
```

Количество знаков может быть числом от 0 до 3. Если количество знаков не указано, предполагается 3.

CURRENT_TRANSACTION

Контекстная переменная `CURRENT_TRANSACTION` типа `INTEGER` возвращает число - системный идентификатор транзакции, под управлением которой выполняется текущий запрос. Значение хранится в странице заголовка базы данных и сбрасывается в 0 после восстановления (или создания базы). Оно увеличивается при старте новой транзакции.

Вряд ли это значение может быть полезно при использовании обычных средств `SQL`.

CURRENT_USER

`CURRENT_USER` типа `VARCHAR(63)` возвращает имя пользователя, который в настоящий момент соединен с базой данных. Возвращаемое значение точно такое же, как и при использовании контекстной переменной `USER`.

DECFLOAT_ROUND

Контекстная переменная `DECFLOAT_ROUND` определяет режим округления, используемый в операциях со значениями `DECFLOAT`.

DECFLOAT_TRAPS

Контекстная переменная `DECFLOAT_TRAPS` определяет условия, приводящие к вызову исключений в операциях со значениями `DECFLOAT`.

INSERTING, UPDATING и DELETING

Контекстные переменные `INSERTING`, `UPDATING` и `DELETING` позволяют определить, какой тип операции с данными базы данных в настоящий момент выполняется. Они возвращают значение `TRUE`, если выполняется, соответственно, оператор добавления новых данных, изменения существующих данных или удаления строк. Эти переменные могут быть использованы только в табличных триггерах.

LOCALTIME

`LOCALTIME` типа `TIME WITHOUT TIME ZONE` возвращает текущее время сервера в часовом поясе сессии. В обращении к контекстной переменной `LOCALTIME` можно указать количество знаков в долях секунды:

```
LOCALTIME [(<количество знаков в долях секунды>)]
```

Количество знаков может быть числом от 0 до 3. Если количество знаков не указано, предполагается 0.

LOCALTIMESTAMP

Контекстная переменная LOCALTIMESTAMP типа TIMESTAMP WITHOUT TIME ZONE возвращает текущую дату и текущее время сервера в часовом поясе сессии. В текущем времени указываются и миллисекунды - три знака после десятичной точки. При обращении к этой контекстной переменной можно задавать требуемое количество долей секунды:

```
LOCALTIMESTAMP [(<количество знаков в долях секунды>)]
```

Количество знаков может быть числом от 0 до 3. Если количество знаков не указано, предполагается 0.

NEW

Контекстная переменная NEW содержит новую версию записи базы данных, которая была вставлена или обновлена. Эта переменная может быть использована только в табличных триггерах.

Эта переменная в триггерах после события (AFTER) доступна только для чтения. Попытка записи в переменную NEW вызовет исключение.

В триггерах на несколько типов событий контекстная переменная NEW доступна всегда. Но при срабатывании триггера по событию DELETE новой версии записи не создаётся, т.е. чтение переменной NEW всегда будет возвращать значение NULL, а запись в неё вызовет исключение времени выполнения.

См. также переменную OLD.

NOW

NOW является не переменной, а строковой константой. Однако, при её приведении (CAST()) к типу даты/времени, результатом будет текущая дата и/или время. Точность составляет до миллисекунды. NOW нечувствительна к регистру и при приведении типов игнорируются начальные и конечные пробелы.

```
select CAST('NOW' as DATE) from rdb$database;
-------------------------------------------
2024-07-03

select CAST('now' as TIMESTAMP) from rdb$database;
-------------------------------------------
2024-07-03 12:45:19.4567
```

При использовании приведения типов NOW всегда возвращает фактическую дату/время, даже в модулях PSQL, где CURRENT_DATE, CURRENT_TIME и CURRENT_TIMESTAMP возвращают одно и то же значение вплоть до окончания выполнения кода. Это делает NOW полезным для измерения временных интервалов в триггерах, процедурах и исполнимых блоках;

См. также переменные CURRENT_DATE, CURRENT_TIME, CURRENT_TIMESTAMP, TODAY.

OLD

Контекстная переменная OLD содержит существующую версию записи базы данных перед удалением или обновлением. Эта переменная доступна только для чтения и только коде триггеров. Тип значения совпадает с типом данных строки.

В триггерах на несколько типов событий контекстная переменная OLD доступна всегда. Очевидно, что при срабатывании триггера по событию INSERT нет никакой существующей ранее версии записи. В этой ситуации переменная OLD всегда будет возвращать NULL; попытка записи в неё вызовет исключение на этапе выполнения.

См. также переменную NEW.

ROW_COUNT

Контекстная переменная ROW_COUNT содержит количество строк, затронутых (были прочитаны, добавлены, изменены или удалены) последним оператором DML (INSERT, UPDATE, DELETE, SELECT или FETCH) в текущем модуле PSQL.

ROW_COUNT нельзя использовать для определения количества строк, затронутых оператором EXECUTE STATEMENT или EXECUTE PROCEDURE.

Поведение при выполнении SELECT и FETCH:

- После однократного SELECT значение ROW_COUNT равно 1, если был получен какой-либо набор данных, и 0 в противном случае.
- В цикле FOR SELECT значение ROW_COUNT увеличивается с каждой итерацией (начиная с 0 перед первым циклом).
- После FETCH из курсора ROW_COUNT равен 1, если была получена строка данных, и 0 в противном случае. Получение большего количества записей из одного и того же курсора не делает ROW_COUNT больше 1.

Пример:

```
update Figures set Number = 0 where id = :id;
if (row_count = 0) then
        insert into Figures (id, Number) values (:id, 0);
```

REPLICATION_STATUS_JSON

Контекстная переменная REPLICATION_STATUS_JSON содержит сведения о статусе репликации в формате JSON:

```
[
{
    "type": 2,
    "connection_string": "/opt/databases/employee.fdb",
    "active": 3,
    "last_modified": 1765806226,
    "waitflush_count": 0,
    "waitflush_time": 0,
    "waitflush_transfer": 0,
    "background_count": 0,
    "background_time": 0,
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
    "background_transfer": 0,
    "server_pid": 19528
},
{
                <сведения о другой реплике>
}
]
```

Сведения о статусе репликации базы данных:

- `type` - тип состояния базы данных:
  - 1 – ведущая БД с синхронной репликацией;
  - 2 – ведущая БД с асинхронной репликацией.
- `connection_string` - строка подключения:
- `active` - статус активности подключения:
  - -2 - реплика отключена после ошибки (для ведущей БД с адаптивной репликацией);
  - -1 - данные на слейве не актуальны (для ведущей БД с адаптивной репликацией);
  - 0 – отключена (для ведущей БД с синхронной репликацией);
  - 1 – подключена (для ведущей БД с синхронной репликацией);
  - <номер_файла> – номер текущего файла журнала (для ведущей БД с асинхронной репликацией).
- `last_modified` - время последней успешной отправки буфера на реплику в формате Unix.
- `waitflush_count` - количество отправленных пакетов (с задержкой ведущей БД, при коммитах).
- `waitflush_time` - количество времени ведущей БД для отправки пакетов (в микросекундах).
- `waitflush_transfer` - размер отправленных данных реплике (в байтах).
- `background_count` - количество отправленных пакетов (в фоне, в рамках одной транзакции).
- `background_time` - количество времени ведущей БД для отправки пакетов (в фоне, в рамках одной транзакции).
- `background_transfer` - размер отправленных данных реплике (в фоне, в рамках одной транзакции).
- `server_pid` - идентификатор серверного процесса.

SQLCODE, GDSCODE

Контекстные переменные `SQLCODE` и `GDSCODE` типа `INTEGER` позволяют получить значения соответствующих кодов ошибок базы данных. Могут использоваться только в хранимых процедурах или триггерах в блоках обработки ошибок базы данных `WHEN`. За пределами таких блоков эти переменные имеют нулевое значение.

SQLSTATE

В `SQL-2003` механизмом, передающим информацию об ошибке, является контекстная переменная состояния `SQLSTATE`. Она представляет собой строку из пяти символов, в которой могут находиться буквы латинского алфавита в верхнем регистре и цифры. Эта строка делится на две группы: двухсимвольный код класса и трехсимвольный код подкласса. Вне обработчиков ошибок переменная `SQLSTATE` равна `00000`, а вне `PSQL` не существует вообще.

`SQLSTATE` предназначен для замены `SQLCODE`.

Если в параметре SQLCODE код класса равен 00, оператор завершился успешно, если 01 - оператор вывел предупреждение, а 02 означает, что нет данных. Любое другое значение кода класса означает, что выполнение оператора не было успешным. Поскольку классы 00, 01 и 02 не вызывают ошибку, они никогда не будут обнаруживаться в переменной SQLSTATE.

```
WHEN ANY DO
BEGIN
    ERR = CASE SQLSTATE
       WHEN '2207' THEN 'Неверный формат даты и времени!'
       WHEN '3D001' THEN 'Имя каталога не найдено!'
       WHEN '42S22' THEN 'Столбец не найден!'
       WHEN '22012' THEN 'Деление на ноль!'
       ELSE 'Есть ошибка!'
    END;
    EXCEPTION EX_EXAMPLE ERR;
END
```

TODAY

Строковая константа TODAY типа CHAR(5). При её приведении (CAST()) к типу даты/времени, результатом будет текущая дата. Написание 'TODAY' нечувствительна к регистру и при приведении типов игнорируются начальные и конечные пробелы.

См. также NOW, TOMORROW, YESTERDAY.

TOMORROW

Строковая константа TOMORROW типа CHAR(8). При её приведении (CAST()) к типу даты/времени, результатом будет дата, следующая за текущей. Написание 'TOMORROW' нечувствительна к регистру и при приведении типов игнорируются начальные и конечные пробелы.

См. также NOW, TODAY, YESTERDAY.

USER

USER - имя пользователя, связанного с текущим экземпляром клиентской библиотеки. Тип данных: VARCHAR(63). Это имя того же самого пользователя, которое может быть получено и при обращении к контекстной переменной CURRENT_USER.

YESTERDAY

Строковая константа YESTERDAY типа CHAR(9). При её приведении (CAST()) к типу даты/времени, результатом будет дата, которая была днем ранее. Написание 'YESTERDAY' нечувствительна к регистру и при приведении типов игнорируются начальные и конечные пробелы.

См. также NOW, TODAY, TOMORROW.

Приложение K Операторы языка хранимых процедур, функций и триггеров

BREAK

Оператор BREAK осуществляет выход из цикла WHILE или FOR. Код продолжает выполняться с первого оператора после прерванного цикла. Синтаксис оператора:

Листинг K.1. Синтаксис оператора BREAK

```
[<метка>:]
<оператор цикла>
BEGIN
     ...
     BREAK;
     ...
END

<оператор цикла> ::=
    FOR <оператор SELECT> INTO <список переменных> DO
  | FOR EXECUTE STATEMENT ... INTO <список переменных> DO
  | WHILE (<логическое условие>) DO
```

CLOSE

Закрывает набор данных, связанный с данным курсором.

Листинг K.2. Синтаксис оператора CLOSE

```
CLOSE <имя курсора>;
```

Курсор с таким именем должен быть предварительно объявлен с помощью оператора DECLARE CURSOR.

См. также операторы FETCH, OPEN, DECLARE CURSOR.

CONTINUE

Оператор CONTINUE моментально начинает новую итерацию внутреннего цикла операторов WHILE или FOR. С использованием опционального параметра метки CONTINUE также может начинать новую итерацию для внешних циклов. Синтаксис оператора:

Листинг K.3. Синтаксис оператора CONTINUE

```
CONTINUE [<метка>];
```

См. также операторы EXIT, LEAVE.

DECLARE CURSOR

Для объявления локальной переменной - курсора используется следующий вариант синтаксиса оператора DECLARE VARIABLE:

Листинг К.4. Синтаксис оператора объявления курсора DECLARE VARIABLE

```
DECLARE [VARIABLE] <имя курсора>
   [SCROLL | NO SCROLL] CURSOR
     FOR (<оператор SELECT>);
```

Этот оператор объявляет именованный курсор, связывая его с набором данных, полученным в операторе SELECT, указанном в предложении CURSOR FOR. В дальнейшем курсор может быть открыт, использоваться для обхода результирующего набора данных, и снова быть закрытым. Также поддерживаются позиционированные обновления и удаления при использовании WHERE CURRENT OF в операторах UPDATE и DELETE. Имя курсора можно использовать в качестве ссылки на курсор, как на переменные типа запись. Текущая запись доступна через имя курсора, что делает необязательным предложение INTO в операторе FETCH.

Курсор может быть однонаправленными прокручиваемым. Необязательное предложение SCROLL делает курсор двунаправленным (прокручиваемым), предложение NO SCROLL - однонаправленным. По умолчанию курсоры являются однонаправленными. Однонаправленные курсоры позволяют двигаться по набору данных только вперёд. Двунаправленные курсоры позволяют двигаться по набору данных не только вперёд, но и назад, а также на N позиций относительно текущего положения.

Особенности использования курсора:

- Предложение FOR UPDATE разрешено использовать в операторе SELECT, но оно не требуется для успешного выполнения позиционированного обновления или удаления.
- Удостоверьтесь, что объявленные имена курсоров не совпадают, ни с какими именами, определёнными позже в предложениях AS CURSOR.
- Если курсор требуется только для прохода по результирующему набору данных, то практически всегда проще (и менее подвержено ошибкам) использовать оператор FOR SELECT с предложением AS CURSOR. Объявленные курсоры должны быть явно открыты, использованы для выборки данных и закрыты. Кроме того, вы должны проверить контекстную переменную ROW_COUNT после каждой выборки и выйти из цикла, если её значение ноль. Предложение FOR SELECT делает эту проверку автоматически. Однако объявленные курсоры дают большие возможности для контроля над последовательными событиями и позволяют управлять несколькими курсорами параллельно.
- Оператор SELECT может содержать параметры, например: SELECT NAME || :SFX FROM NAMES WHERE NUMBER = :NUM. Каждый параметр должен быть заранее объявлен как переменная PSQL (это касается также входных и выходных параметров). При открытии курсора параметру присваивается текущее значение переменной.
- Если опция прокрутки опущена, то по умолчанию принимается NO SCROLL (т.е курсор открыт для движения только вперёд). Это означает, что могут быть использованы только команды FETCH [NEXT FROM]. Другие команды будут возвращать ошибки.

См. также операторы OPEN, CLOSE, FETCH, DECLARE VARIABLE.

DECLARE FUNCTION

В хранимых функциях и хранимых процедурах можно объявлять подфункции. Синтаксис оператора:

Листинг К.5. Синтаксис оператора объявления подфункции DECLARE FUNCTION

```
DECLARE FUNCTION <имя подфункции>
     [(<входной параметр> [, <входной параметр> ...])]
RETURNS <тип> [COLLATE <сортировка>] [DETERMINISTIC]
AS
     [<объявление лок.переменных/курсоров>[<объявление лок.переменных/курсоров>...] ]
BEGIN
     <блок операторов>
END
```

Подфункция не может быть вложена в другую подпрограмму. Они поддерживаются только в основном модуле (хранимой процедуре, хранимой функции и анонимном PSQL блоке).

В настоящее время подфункция не имеет прямого доступа для использования переменных, курсоров и других подпрограмм из основного модуля. Это может быть разрешено в будущем.

Одна подпрограмма может вызывать и другую подпрограмму, в том числе рекурсивно. В ряде случаев может потребоваться предварительное объявление подпрограммы. Общее правило: одна подпрограмма может вызвать другую подпрограмму, если последняя объявлена выше точки вызова. Все объявленные подпрограммы должны быть реализованы с той же сигнатурой. Значения по умолчанию для параметров подпрограмм не могут быть переопределены. Это означает, что они могут быть определены в реализации только тех подпрограмм, которые не были объявлены ранее

См. также операторы DECLARE VARIABLE, DECLARE PROCEDURE.

DECLARE PROCEDURE

В хранимых функциях и хранимых процедурах можно объявлять подпроцедуры. Синтаксис оператора:

Листинг К.6. Синтаксис оператора объявления подпроцедуры DECLARE PROCEDURE

```
DECLARE PROCEDURE <имя подпроцедуры>
     [(<входной параметр> [, <входной параметр> ...])]
[RETURNS (<выходной параметр> [, <выходной параметр> ...])]
AS
     [<объявление лок.переменных/курсоров>[<объявление лок.переменных/курсоров>...] ]
BEGIN
     <блок операторов>
END
```

Подпроцедура не может быть вложена в другую подпрограмму. Они поддерживаются только в основном модуле (хранимой процедуре, хранимой функции и анонимном PSQL блоке). В настоящее время подпроцедура не имеет прямого доступа для использования переменных, курсоров и других подпрограмм из основного модуля. Это может быть разрешено в будущем. Одна подпрограмма может вызывать и другую подпрограмму, в том числе рекурсивно. В ряде случаев может потребоваться предварительное объявление подпрограммы. Общее правило: одна подпрограмма может вызвать другую подпрограмму, если последняя объявлена выше точки вызова. Все объявленные подпрограммы должны быть реализованы с той же сигнатурой. Значения по умолчанию для параметров подпрограмм не могут быть переопределены. Это означает, что они могут быть определены в реализации только тех подпрограмм, которые не были объявлены ранее.

См. также операторы DECLARE VARIABLE, DECLARE FUNCTION.

DECLARE VARIABLE

Оператор позволяет определить локальную переменную или курсор.

Листинг К.7. Синтаксис оператора объявления подпроцедуры DECLARE VARIABLE

```
DECLARE [VARIABLE] {
  <имя локальной переменной> <тип>
        [NOT NULL]
        [COLLATE <порядок сортировки>]
        [{ = | DEFAULT } <значение по умолчанию>]
| <имя курсора> [SCROLL | NO SCROLL] CURSOR FOR (<оператор SELECT>) }

<тип> ::= {
  <тип данных SQL>
| [TYPE OF] <имя домена>
| TYPE OF COLUMN <имя таблицы/представления>.<имя столбца> }

<значение по умолчанию>::= {<литерал> | NULL | <контекстная переменная>}
```

Имя переменной должно быть уникальным среди всех имен локальных переменных, имен входных и выходных параметров данного программного объекта.

Типом данных может быть любой тип данных, используемый в SQL.

Вместо типа данных можно указать имя домена. В этом случае переменной присваиваются все характеристики домена — запрет пустого значения (NOT NULL), значение по умолчанию (DEFAULT) и условие (CHECK), которому должно удовлетворять значение, помещаемое в переменную.

В случае задания в операторе предложения TYPE OF для этой переменной из домена копируется лишь тип данных.

Локальные переменные можно объявлять, используя тип данных столбцов существующих таблиц и представлений. Для этого используется предложение TYPE OF COLUMN, после которого указывается имя таблиц или представления и через точку имя столбца. При использовании TYPE OF COLUMN наследуется только тип данных, а в случае строковых типов ещё набор символов и порядок сортировки. Ограничения и значения по умолчанию столбца никогда не используются.

Для локальных переменных можно указать ограничение NOT NULL, тем самым запретив передавать в него значение NULL.

Для строкового типа данных также можно задать порядок сортировки (предложение COLLATE).

Локальной переменной можно устанавливать инициализирующее (начальное) значение. Это значение устанавливается с помощью предложения DEFAULT или оператора =. В качестве значения по умолчанию может быть использовано значение NULL, литерал и любая контекстная переменная совместимая по типу данных.

При помощи оператора DECLARE VARIABLE также можно объявить курсор - специфическую переменную, связанную с выбираемым из базы данных набором данных. Получаемый набор данных определяется оператором SELECT, который следует после ключевых слов CURSOR FOR и заключается в круглые скобки.

См. также операторы OPEN, CLOSE, FETCH, DECLARE CURSOR.

EXCEPTION

Выдает указанное пользовательское исключение базы данных. При возбуждении исключения можно также указать альтернативный текст сообщения, который заменит текст сообщения заданным при создании исключения.

Листинг К.8. Синтаксис оператора EXCEPTION

```
EXCEPTION <имя пользовательского исключения> [<текст сообщения> | USING (<значение> [,
<значение>...])];
```

Оператор EXCEPTION возбуждает пользовательское исключение с указанным именем. При возбуждении исключения можно также указать альтернативный текст сообщения, который заменит текст сообщения заданным при создании исключения.

Текст сообщения исключения может содержать слоты для параметров, которые заполняются при возбуждении исключения. Для передачи значений параметров в исключение используется предложение USING. Параметры рассматриваются слева направо. Каждый параметр передаётся в оператор возбуждающий исключение как N-ый, N начинается с 1:

- Если N-ый параметр не передан, его слот не заменяется;
- Если передано значение NULL, слот будет заменён на строку '***null***';
- Если количество передаваемых параметров будет больше, чем содержится в сообщении исключения, то лишние будут проигнорированы;
- Максимальный номер параметра равен 9;
- Общая длина сообщения, включая значения параметров, ограничена 1053 байтами.

```
CREATE EXCEPTION ex1 'something wrong in @1';
set term !;
EXECUTE BLOCK AS
BEGIN
   EXCEPTION ex1 USING ('the text');
END!
set term ;!
```

Исключение может быть обработано в операторе WHEN-DO. Если пользовательское исключение не было обработано в триггере или в хранимой процедуре или функции, то все выполненные действия отменяются, вызвавшая программа получает текст, заданный при создании исключения.

См. также операторы CREATE EXCEPTION, ALTER EXCEPTION, DROP EXCEPTION, WHEN-DO.

EXIT

Оператор EXIT позволяет из любой точки триггера или хранимой процедуры или функции перейти на конечный оператор END, то есть завершить выполнение программы.

Листинг К.9. Синтаксис оператора EXIT

```
EXIT;
```

См. также операторы LEAVE, SUSPEND, CONTINUE.

FETCH

Оператор FETCH читает очередную запись набора данных, связанного с курсором. Синтаксис оператора:

Листинг К.10. Синтаксис оператора FETCH

```
FETCH <имя курсора>
     [INTO :<внутренняя переменная> [, :<внутренняя переменная>]... ];

FETCH {
  NEXT
| PRIOR
| FIRST
| LAST
| ABSOLUTE <n>
| RELATIVE <n>
} FROM <имя курсора> [INTO [:]<внутр. переменная> [,[:]<внутр. переменная>...]];
```

Оператор FETCH применим только к курсорам, объявленным в операторе DECLARE VARIABLE.

Оператор читает очередную строку, полученную при выполнении оператора SELECT, связанного с данным курсором, и помещает полученные данные во внутренние переменные программы (предложение INTO). Предложение INTO можно не указывать в том случае, если для полученной строки будет использован оператор удаления данных DELETE.

В новой версии оператора FETCH можно указывать в каком направлении и на сколько записей продвинется позиция курсора.

Предложение NEXT указывает, что указатель курсора должен продвинуться на 1 запись вперёд. Это предложение допустимо использовать как с прокручиваемыми, там и не прокручиваемыми курсорами. Остальные предложения допустимо использовать только с прокручиваемыми курсорами. Предложение PRIOR указывает, что указатель курсора должен продвинуться на 1 запись назад. Предложение FIRST позволяет переместить позицию курсора на первую запись, а предложение LAST - на последнюю. Предложение ABSOLUTE позволяет указать номер позиции, на которую будет установлен курсор. Номер позиции должен быть в диапазоне от 1 до максимального количества записей извлекаемых запросом курсора. Предложение RELATIVE позволяет указать, на какое количество записей относительно текущей позиции необходимо переместить указатель курсора. Если указано положительное число, то курсор перемещает вперёд на N позиций, если отрицательное, то назад.

Позволяется использовать ссылки на курсоры, как на переменные типа запись. Текущая запись доступна через имя курсора.

- Для разрешения неоднозначности при доступе к переменной курсора перед именем курсора необходим префикс двоеточие;
- К переменной курсора можно получить доступ без префикса двоеточия, но в этом случае, в зависимости от области видимости контекстов, существующих в запросе, имя может разрешиться как контекст запроса вместо курсора;
- Переменные курсора доступны только для чтения;
- Чтение из переменной курсора возвращает текущие значения полей. Это означает, что оператор UPDATE (с предложением WHERE CURRENT OF) обновит также и значения полей в переменной курсора для последующих чтений. Выполнение оператора DELETE (с предложением WHERE CURRENT OF) установит NULL для значений полей переменной курсора для последующих чтений.

Для проверки того, что записи набора данных исчерпаны, используется контекстная переменная ROW_COUNT, которая возвращает количество считанных оператором строк. Если произошло чтение очередной записи из набора данных, то ROW_COUNT равняется единице, иначе нулю.

См. также операторы OPEN, CLOSE, DECLARE VARIABLE, WHEN-DO, описание контекстной переменной ROW_COUNT.

FOR EXECUTE STATEMENT

Оператор FOR EXECUTE STATEMENT является оператором цикла. Синтаксис оператора представлен в следующем листинге:

Листинг К.11. Синтаксис оператора FOR EXECUTE STATEMENT

```
[<метка>:]
[FOR] EXECUTE STATEMENT <строковое выражение>
     [ON EXTERNAL [DATA SOURCE] <спецификация файла> ]
     [WITH {AUTONOMOUS | COMMON } TRANSACTION ]
     [AS USER <имя пользователя>]
     [PASSWORD <пароль>]
     [CERTIFICATE <алиас сертификата>]
     [PIN <пароль>]
     [ROLE <роль>]
     [WITH CALLER PRIVILEGES]
     [INTO [:] <внутренняя переменная> [, [:] <внутренняя переменная>...] ]
[DO <составной оператор>]

<строковое выражение> ::= {
  <Строки или переменная, содержащая не параметризованный SQL запрос>
| (<Строки или переменная, содержащая не параметризованный SQL запрос>)
| (<Строки или переменная, содержащая параметризованный SQL запрос>) ({ <именованные
параметры> | <позиционные параметры> }) }

<именованные параметры> ::= [EXCESS] <имя параметра>:=<выражение>
                                                    [, [EXCESS] <имя параметра>:=
<выражение> ...]

<позиционные параметры> ::= <выражение> [, <выражение> ...]

<спецификация файла> ::=
     [<спецификация удалённого сервера>] { <путь к файлу БД> | <алиас БД> }

<спецификация удалённого сервера> ::=
  <хост>[\<порт> | <имя сервиса>]:
| \\<хост>[@<порт> | <имя сервиса>]\
| <протокол>://[ <имя сервиса>[: <порт> | <имя сервиса>]/]

<протокол> = inet | inet4 | inet6 | xnet
```

В этом операторе "строковое выражение" — любое выражение, возвращающее строку символов. Выражение может быть внутренней переменной, значением, получаемым конкатенацией двух или более строк, строковым литералом, заключенным в апострофы, и т.д. Содержанием этого выражения должен быть правильный оператор SELECT, обращающийся к таблице, представлению или к хранимой процедуре выбора для получения данных.

Данные, полученные из оператора SELECT, при помощи обязательного предложения INTO помещаются во внутренние переменные. Именам внутренних переменных в этом предложении должны предшествовать символы двоеточия.

Для каждой считанной записи выполняется составной оператор, указанный после ключевого слова DO. Цикл повторяется, пока не будут прочитаны все строки или пока не встретится оператор LEAVE. После этого происходит выход из цикла.

Более подробно об операторе EXECUTE STATEMENT изложено в разделе 17.16 "Оператор FOR EXECUTE STATEMENT". Пример использования оператора FOR EXECUTE STATEMENT представлен в примере, в котором выполняются те же действия, что и в операторе FOR SELECT-DO:

```
DECLARE VARIABLE STMT1 CHAR(100);
...
STMT1 = 'SELECT CODCOUNTRY, CODREGION, NAMEREG, CENTER FROM VIEW_RUSSIA2';
FOR EXECUTE STATEMENT STMT1
INTO :CODCOUNTRY, :CODREGION, :NAMEREG, :CENTER
DO
   BEGIN
      EXECUTE PROCEDURE PROC_N(CODCOUNTRY, CODREGION, NAMEREG, CENTER)
      RETURNING_VALUES RESULT;
      SUSPEND;
   END
```

Здесь для формирования оператора SELECT используется локальная переменная STMT1 строкового типа. Ей присваивается соответствующее значение. После этого выполняется оператор FOR EXECUTE STATEMENT, который из представления VIEW_RUSSIA2 читает очередную строку таблицы регионов. Данные прочитанной строки обрабатываются в процедуре PROC_N, после чего происходит временная приостановка выполнения процедуры и управление передается вызвавшей программе.

Когда будут прочитаны все строки, соответствующие условиям поиска в представлении VIEW_-RUSSIA2, выполнение цикла будет завершено и управление перейдет к оператору, следующему за оператором цикла.

См. также операторы LEAVE, SUSPEND, EXIT, FOR SELECT-DO, WHILE-DO, WHEN-DO, EXECUTE BLOCK.

FOR SELECT-DO

Оператор FOR SELECT-DO является оператором цикла. Синтаксис оператора:

Листинг К.12. Синтаксис оператора Синтаксис оператора FOR SELECT

```
[<метка>:]
FOR
     <оператор SELECT>
     [AS CURSOR <имя курсора>]
     INTO [:]<имя переменной/параметра> [, [:]<имя переменной/параметра> ...]
DO <составной оператор>;
```

Оператор SELECT выбирает очередную строку из таблицы (представления, хранимой процедуры выбора), после чего выполняется составной оператор. Оператор SELECT должен содержать предложение INTO, которое располагается в конце этого оператора. Цикл повторяется, пока не будут прочитаны все строки. После этого происходит выход из цикла. Цикл также может быть завершен до прочтения всех строк при использовании оператора LEAVE.

Необязательное предложение AS CURSOR создаёт именованный курсор, на который можно ссылаться (с использованием предложения WHERE CURRENT OF) внутри оператора или блока операторов следующего после предложения DO, для того чтобы удалить или модифицировать текущую строку. Над курсором, объявленным с помощью предложения AS CURSOR нельзя выполнять операторы OPEN, FETCH и CLOSE.

См. также операторы LEAVE, SUSPEND, EXIT, FOR EXECUTE STATEMENT, WHILE-DO, WHILE-DO, EXECUTE BLOCK.

IF-THEN-ELSE

Листинг К.13. Синтаксис оператора IF-THEN-ELSE

```
IF (<условие>)
THEN <составной оператор>
[ELSE <составной оператор>];
```

Условием является обычное условие, которое может возвращать значения TRUE, FALSE или UNKNOWN. Если условие возвращает значение TRUE, то выполняется составной оператор после ключевого слова THEN. Иначе (если условие возвращает FALSE или UNKNOWN) выполняется составной оператор после ключевого слова ELSE, если присутствует. Условие всегда заключается в круглые скобки.

Составной оператор — это одиночный оператор или блок операторов, заключенных в операторные скобки BEGIN и END.

См. также внутренние функции IIF, CASE-WHEN-ELSE, COALESCE.

LEAVE

Этот оператор выполняет выход из цикла WHILE или FOR независимо от выполнения условия в предложении WHILE. Если же в операторе указана метка, то выполняется переход на начало другого (или того же самого) цикла. Синтаксис оператора:

Листинг К.14. Синтаксис оператора LEAVE

```
LEAVE [<метка>];
```

Если в операторе не указана метка, то оператор просто осуществляет выход из текущего цикла WHILE.

Если в операторе LEAVE указана метка, то это должна быть метка, относящаяся к оператору WHILE, к оператору FOR SELECT или к оператору FOR EXECUTE STATEMENT. При выполнении такого оператора LEAVE происходит переход к выполнению соответствующего циклического оператора.

См. также оператор EXIT, CONTINUE.

OPEN

Оператор открывает курсор (читает данные), связанный с оператором SELECT, выбирающим данные из таблицы базы данных или из представления. Синтаксис оператора:

Листинг К.15. Синтаксис оператора OPEN

```
OPEN <имя курсора>;
```

Оператор OPEN применим только к курсорам, объявленным в операторе DECLARE CURSOR.

См. также операторы FETCH, CLOSE, WHEN-DO.

POST_EVENT

Оператор посылает событие (сообщение) всем клиентским приложениям, подключенным в настоящий момент к базе данных, и «прослушивающим» данное событие.

Листинг К.16. Синтаксис оператора POST_EVENT

```
POST_EVENT {
  '<имя сообщения>'
| <имя столбца>
| :<внутренняя переменная> };
```

Это может быть явно заданный текст в виде строки символов, имя столбца, содержащего текст, или имя внутренней переменной (локальной переменной, входного или выходного параметра хранимой процедуры), которая содержит отправляемый текст сообщения. Имя события, ограничено 127 байтами.

См. также операторы CREATE EXCEPTION, ALTER EXCEPTION, EXCEPTION, DROP EXCEPTION, WHEN-DO.

SUSPEND

Оператор временно приостанавливает выполнение хранимой процедуры выбора и передает вызвавшей программе значения выходных параметров. Когда вызвавшая программа выполняет после этого оператор FETCH (этот оператор неявно выдается при выполнении оператора SELECT), работа процедуры возобновляется с оператора, следующего непосредственно за оператором SUSPEND.

Листинг К.17. Синтаксис оператора SUSPEND

```
SUSPEND;
```

Если оператор SUSPEND выдается в выполняемой хранимой процедуре, то это равносильно выполнению оператора EXIT, в результате чего завершается работа процедуры.

См. также операторы LEAVE, EXIT.

WHEN-DO

Оператор используется для обработки ошибочных ситуаций и пользовательских исключений в языке хранимых процедур, функций и триггеров. Оператор позволяет перехватить любые указанные ошибки базы данных и/или пользовательские исключения или вообще все возникшие ошибочные ситуации или выданные пользовательские исключения (EXCEPTION) при обращении к базе данных.

Листинг К.18. Синтаксис оператора WHEN-DO

```
WHEN { <ошибка> [, <ошибка> ...] | ANY }
DO <составной оператор>;

<ошибка> ::= {
  SQLCODE <код ошибки SQLCODE>
| SQLSTATE <код ошибки SQLSTATE>
| GDSCODE <код ошибки GDSCODE>
| EXCEPTION <имя пользовательского исключения> }
```

Оператор должен находиться в самом конце блока операторов перед оператором END. В условии оператора, до ключевого слова DO, задаются те ситуации, при которых будет выполняться составной оператор. Здесь можно перечислить произвольное количество значений кодов SQLCODE, GDSCODE, имен пользовательских исключений или задать ключевое слово ANY, которое означает, что обработка ситуации будет выполняться при появлении любой ошибки базы данных и/или любого пользовательского исключения.

После ключевого слова DO следует оператор или блок операторов, заключенных в операторные скобки BEGIN и END, которые выполняют некоторую обработку возникшей ситуации.

Оператор WHEN-DO вызывается только в том случае, если произошло одно из указанных в его условии событий. В случае выполнения оператора (даже если в нем фактически не было выполнено никаких действий) ошибка или пользовательское исключение не прерывает и не отменяет действий триггера или хранимой процедуры или функции, где был выдан этот оператор, работа продолжается, как если бы никаких исключительных ситуаций не было. Оператор перехватывает ошибки и исключения в текущем блоке операторов. Он также перехватывает подобные ситуации во вложенных блоках, если эти ситуации не были в них обработаны.

См. также операторы FOR SELECT-DO, FETCH, EXCEPTION, WHILE-DO, FOR EXECUTE STATEMENT, EXECUTE BLOCK.

WHILE-DO

Оператор позволяет организовать в PSQL цикл. Синтаксис оператора представлен в следующем листинге:

Листинг К.19. Синтаксис оператора WHILE-DO

```
[<метка>:]
WHILE (<условие>) DO
    <составной оператор>
```

Составной оператор будет выполняться в цикле, пока условие возвращает значение TRUE. Циклы могут быть вложенными, глубина вложения не ограничена.

См. также операторы IF-THEN-ELSE, LEAVE, FOR SELECT-DO, WHEN-DO, EXECUTE BLOCK, внутренние функции IIF, CASE-WHEN-ELSE, COALESCE.

Приложение Л Список региональных часовых поясов

Таблица Л.1 — Список региональных часовых поясов и их идентификаторов

Часовой пояс	ID	Часовой пояс	ID
GMT	65535	ACT	65534
AET	65533	AGT	65532
ART	65531	AST	65530
Africa/Abidjan	65529	Africa/Accra	65528
Africa/Addis_Ababa	65527	Africa/Algiers	65526
Africa/Asmara	65525	Africa/Asmera	65524
Africa/Bamako	65523	Africa/Bangui	65522
Africa/Banjul	65521	Africa/Bissau	65520
Africa/Blantyre	65519	Africa/Brazzaville	65518
Africa/Bujumbura	65517	Africa/Cairo	65516
Africa/Casablanca	65515	Africa/Ceuta	65514
Africa/Conakry	65513	Africa/Dakar	65512
Africa/Dar_es_Salaam	65511	Africa/Djibouti	65510
Africa/Douala	65509	Africa/El_Aaiun	65508
Africa/Freetown	65507	Africa/Gaborone	65506
Africa/Harare	65505	Africa/Johannesburg	65504
Africa/Juba	65503	Africa/Kampala	65502
Africa/Khartoum	65501	Africa/Kigali	65500
Africa/Kinshasa	65499	Africa/Lagos	65498
Africa/Libreville	65497	Africa/Lome	65496
Africa/Luanda	65495	Africa/Lubumbashi	65494
Africa/Lusaka	65493	Africa/Malabo	65492
Africa/Maputo	65491	Africa/Maseru	65490
Africa/Mbabane	65489	Africa/Mogadishu	65488
Africa/Monrovia	65487	Africa/Nairobi	65486
Africa/Ndjamena	65485	Africa/Niamey	65484
Africa/Nouakchott	65483	Africa/Ouagadougou	65482
Africa/Porto-Novo	65481	Africa/Sao_Tome	65480
Africa/Timbuktu	65479	Africa/Tripoli	65478
Africa/Tunis	65477	Africa/Windhoek	65476
America/Adak	65475	America/Anchorage	65474
America/Anguilla	65473	America/Antigua	65472
America/Araguaina	65471	America/Argentina/Buenos_Aires	65470

(разрыв таблицы)

(разрыв таблицы)

Часовой пояс	ID	Часовой пояс	ID
America/Argentina/Catamarca	65469	America/Argentina/ComodRivadavia	65468
America/Argentina/Cordoba	65467	America/Argentina/Jujuy	65466
America/Argentina/La_Rioja	65465	America/Argentina/Mendoza	65464
America/Argentina/Rio_Gallegos	65463	America/Argentina/Salta	65462
America/Argentina/San_Juan	65461	America/Argentina/San_Luis	65460
America/Argentina/Tucuman	65459	America/Argentina/Ushuaia	65458
America/Aruba	65457	America/Asuncion	65456
America/Atikokan	65455	America/Atka	65454
America/Bahia	65453	America/Bahia_Banderas	65452
America/Barbados	65451	America/Belem	65450
America/Belize	65449	America/Blanc-Sablon	65448
America/Boa_Vista	65447	America/Bogota	65446
America/Boise	65445	America/Buenos_Aires	65444
America/Cambridge_Bay	65443	America/Campo_Grande	65442
America/Cancun	65441	America/Caracas	65440
America/Catamarca	65439	America/Cayenne	65438
America/Cayman	65437	America/Chicago	65436
America/Chihuahua	65435	America/Coral_Harbour	65434
America/Cordoba	65433	America/Costa_Rica	65432
America/Creston	65431	America/Cuiaba	65430
America/Curacao	65429	America/Danmarkshavn	65428
America/Dawson	65427	America/Dawson_Creek	65426
America/Denver	65425	America/Detroit	65424
America/Dominica	65423	America/Edmonton	65422
America/Eirunepe	65421	America/El_Salvador	65420
America/Ensenada	65419	America/Fort_Nelson	65418
America/Fort_Wayne	65417	America/Fortaleza	65416
America/Glace_Bay	65415	America/Godthab	65414
America/Goose_Bay	65413	America/Grand_Turk	65412
America/Grenada	65411	America/Guadeloupe	65410
America/Guatemala	65409	America/Guayaquil	65408
America/Guyana	65407	America/Halifax	65406
America/Havana	65405	America/Hermosillo	65404
America/Indiana/Indianapolis	65403	America/Indiana/Knox	65402
America/Indiana/Marengo	65401	America/Indiana/Petersburg	65400
America/Indiana/Tell_City	65399	America/Indiana/Vevay	65398
America/Indiana/Vincennes	65397	America/Indiana/Winamac	65396

(разрыв таблицы)

(разрыв таблицы)

Часовой пояс	ID	Часовой пояс	ID
America/Indianapolis	65395	America/Inuvik	65394
America/Iqaluit	65393	America/Jamaica	65392
America/Jujuy	65391	America/Juneau	65390
America/Kentucky/Louisville	65389	America/Kentucky/Monticello	65388
America/Knox_IN	65387	America/Kralendijk	65386
America/La_Paz	65385	America/Lima	65384
America/Los_Angeles	65383	America/Louisville	65382
America/Lower_Princes	65381	America/Maceio	65380
America/Managua	65379	America/Manaus	65378
America/Marigot	65377	America/Martinique	65376
America/Matamoros	65375	America/Mazatlan	65374
America/Mendoza	65373	America/Menominee	65372
America/Merida	65371	America/Metlakatla	65370
America/Mexico_City	65369	America/Miquelon	65368
America/Moncton	65367	America/Monterrey	65366
America/Montevideo	65365	America/Montreal	65364
America/Montserrat	65363	America/Nassau	65362
America/New_York	65361	America/Nipigon	65360
America/Nome	65359	America/Noronha	65358
America/North_Dakota/Beulah	65357	America/North_Dakota/Center	65356
America/North_Dakota/New_Salem	65355	America/Ojinaga	65354
America/Panama	65353	America/Pangnirtung	65352
America/Paramaribo	65351	America/Phoenix	65350
America/Port-au-Prince	65349	America/Port_of_Spain	65348
America/Porto_Acre	65347	America/Porto_Velho	65346
America/Puerto_Rico	65345	America/Punta_Arenas	65344
America/Rainy_River	65343	America/Rankin_Inlet	65342
America/Recife	65341	America/Regina	65340
America/Resolute	65339	America/Rio_Branco	65338
America/Rosario	65337	America/Santa_Isabel	65336
America/Santarem	65335	America/Santiago	65334
America/Santo_Domingo	65333	America/Sao_Paulo	65332
America/Scoresbysund	65331	America/Shiprock	65330
America/Sitka	65329	America/St_Barthelemy	65328
America/St_Johns	65327	America/St_Kitts	65326
America/St_Lucia	65325	America/St_Thomas	65324
America/St_Vincent	65323	America/Swift_Current	65322

(разрыв таблицы)

(разрыв таблицы)

Часовой пояс	ID	Часовой пояс	ID
America/Tegucigalpa	65321	America/Thule	65320
America/Thunder_Bay	65319	America/Tijuana	65318
America/Toronto	65317	America/Tortola	65316
America/Vancouver	65315	America/Virgin	65314
America/Whitehorse	65313	America/Winnipeg	65312
America/Yakutat	65311	America/Yellowknife	65310
Antarctica/Casey	65309	Antarctica/Davis	65308
Antarctica/DumontDUrville	65307	Antarctica/Macquarie	65306
Antarctica/Mawson	65305	Antarctica/McMurdo	65304
Antarctica/Palmer	65303	Antarctica/Rothera	65302
Antarctica/South_Pole	65301	Antarctica/Syowa	65300
Antarctica/Troll	65299	Antarctica/Vostok	65298
Arctic/Longyearbyen	65297	Asia/Aden	65296
Asia/Almaty	65295	Asia/Amman	65294
Asia/Anadyr	65293	Asia/Aqtau	65292
Asia/Aqtobe	65291	Asia/Ashgabat	65290
Asia/Ashkhabad	65289	Asia/Atyrau	65288
Asia/Baghdad	65287	Asia/Bahrain	65286
Asia/Baku	65285	Asia/Bangkok	65284
Asia/Barnaul	65283	Asia/Beirut	65282
Asia/Bishkek	65281	Asia/Brunei	65280
Asia/Calcutta	65279	Asia/Chita	65278
Asia/Choibalsan	65277	Asia/Chongqing	65276
Asia/Chungking	65275	Asia/Colombo	65274
Asia/Dacca	65273	Asia/Damascus	65272
Asia/Dhaka	65271	Asia/Dili	65270
Asia/Dubai	65269	Asia/Dushanbe	65268
Asia/Famagusta	65267	Asia/Gaza	65266
Asia/Harbin	65265	Asia/Hebron	65264
Asia/Ho_Chi_Minh	65263	Asia/Hong_Kong	65262
Asia/Hovd	65261	Asia/Irkutsk	65260
Asia/Istanbul	65259	Asia/Jakarta	65258
Asia/Jayapura	65257	Asia/Jerusalem	65256
Asia/Kabul	65255	Asia/Kamchatka	65254
Asia/Karachi	65253	Asia/Kashgar	65252
Asia/Kathmandu	65251	Asia/Katmandu	65250
Asia/Khandyga	65249	Asia/Kolkata	65248

(разрыв таблицы)

(разрыв таблицы)

Часовой пояс	ID	Часовой пояс	ID
Asia/Krasnoyarsk	65247	Asia/Kuala_Lumpur	65246
Asia/Kuching	65245	Asia/Kuwait	65244
Asia/Macao	65243	Asia/Macau	65242
Asia/Magadan	65241	Asia/Makassar	65240
Asia/Manila	65239	Asia/Muscat	65238
Asia/Nicosia	65237	Asia/Novokuznetsk	65236
Asia/Novosibirsk	65235	Asia/Omsk	65234
Asia/Oral	65233	Asia/Phnom_Penh	65232
Asia/Pontianak	65231	Asia/Pyongyang	65230
Asia/Qatar	65229	Asia/Qyzylorda	65228
Asia/Rangoon	65227	Asia/Riyadh	65226
Asia/Saigon	65225	Asia/Sakhalin	65224
Asia/Samarkand	65223	Asia/Seoul	65222
Asia/Shanghai	65221	Asia/Singapore	65220
Asia/Srednekolymsk	65219	Asia/Taipei	65218
Asia/Tashkent	65217	Asia/Tbilisi	65216
Asia/Tehran	65215	Asia/Tel_Aviv	65214
Asia/Thimbu	65213	Asia/Thimphu	65212
Asia/Tokyo	65211	Asia/Tomsk	65210
Asia/Ujung_Pandang	65209	Asia/Ulaanbaatar	65208
Asia/Ulan_Bator	65207	Asia/Urumqi	65206
Asia/Ust-Nera	65205	Asia/Vientiane	65204
Asia/Vladivostok	65203	Asia/Yakutsk	65202
Asia/Yangon	65201	Asia/Yekaterinburg	65200
Asia/Yerevan	65199	Atlantic/Azores	65198
Atlantic/Bermuda	65197	Atlantic/Canary	65196
Atlantic/Cape_Verde	65195	Atlantic/Faeroe	65194
Atlantic/Faroe	65193	Atlantic/Jan_Mayen	65192
Atlantic/Madeira	65191	Atlantic/Reykjavik	65190
Atlantic/South_Georgia	65189	Atlantic/St_Helena	65188
Atlantic/Stanley	65187	Australia/ACT	65186
Australia/Adelaide	65185	Australia/Brisbane	65184
Australia/Broken_Hill	65183	Australia/Canberra	65182
Australia/Currie	65181	Australia/Darwin	65180
Australia/Eucla	65179	Australia/Hobart	65178
Australia/LHI	65177	Australia/Lindeman	65176
Australia/Lord_Howe	65175	Australia/Melbourne	65174

(разрыв таблицы)

(разрыв таблицы)

Часовой пояс	ID	Часовой пояс	ID
Australia/NSW	65173	Australia/North	65172
Australia/Perth	65171	Australia/Queensland	65170
Australia/South	65169	Australia/Sydney	65168
Australia/Tasmania	65167	Australia/Victoria	65166
Australia/West	65165	Australia/Yancowinna	65164
BET	65163	BST	65162
Brazil/Acre	65161	Brazil/DeNoronha	65160
Brazil/East	65159	Brazil/West	65158
CAT	65157	CET	65156
CNT	65155	CST	65154
CST6CDT	65153	CTT	65152
Canada/Atlantic	65151	Canada/Central	65150
Canada/East-Saskatchewan	65149	Canada/Eastern	65148
Canada/Mountain	65147	Canada/Newfoundland	65146
Canada/Pacific	65145	Canada/Saskatchewan	65144
Canada/Yukon	65143	Chile/Continental	65142
Chile/EasterIsland	65141	Cuba	65140
EAT	65139	ECT	65138
EET	65137	EST	65136
EST5EDT	65135	Egypt	65134
Eire	65133	Etc/GMT	65132
Etc/GMT+0	65131	Etc/GMT+1	65130
Etc/GMT+10	65129	Etc/GMT+11	65128
Etc/GMT+12	65127	Etc/GMT+2	65126
Etc/GMT+3	65125	Etc/GMT+4	65124
Etc/GMT+5	65123	Etc/GMT+6	65122
Etc/GMT+7	65121	Etc/GMT+8	65120
Etc/GMT+9	65119	Etc/GMT-0	65118
Etc/GMT-1	65117	Etc/GMT-10	65116
Etc/GMT-11	65115	Etc/GMT-12	65114
Etc/GMT-13	65113	Etc/GMT-14	65112
Etc/GMT-2	65111	Etc/GMT-3	65110
Etc/GMT-4	65109	Etc/GMT-5	65108
Etc/GMT-6	65107	Etc/GMT-7	65106
Etc/GMT-8	65105	Etc/GMT-9	65104
Etc/GMT0	65103	Etc/Greenwich	65102
Etc/UCT	65101	Etc/UTC	65100

(разрыв таблицы)

(разрыв таблицы)

Часовой пояс	ID	Часовой пояс	ID
Etc/Universal	65099	Etc/Zulu	65098
Europe/Amsterdam	65097	Europe/Andorra	65096
Europe/Astrakhan	65095	Europe/Athens	65094
Europe/Belfast	65093	Europe/Belgrade	65092
Europe/Berlin	65091	Europe/Bratislava	65090
Europe/Brussels	65089	Europe/Bucharest	65088
Europe/Budapest	65087	Europe/Busingen	65086
Europe/Chisinau	65085	Europe/Copenhagen	65084
Europe/Dublin	65083	Europe/Gibraltar	65082
Europe/Guernsey	65081	Europe/Helsinki	65080
Europe/Isle_of_Man	65079	Europe/Istanbul	65078
Europe/Jersey	65077	Europe/Kaliningrad	65076
Europe/Kiev	65075	Europe/Kirov	65074
Europe/Lisbon	65073	Europe/Ljubljana	65072
Europe/London	65071	Europe/Luxembourg	65070
Europe/Madrid	65069	Europe/Malta	65068
Europe/Mariehamn	65067	Europe/Minsk	65066
Europe/Monaco	65065	Europe/Moscow	65064
Europe/Nicosia	65063	Europe/Oslo	65062
Europe/Paris	65061	Europe/Podgorica	65060
Europe/Prague	65059	Europe/Riga	65058
Europe/Rome	65057	Europe/Samara	65056
Europe/San_Marino	65055	Europe/Sarajevo	65054
Europe/Saratov	65053	Europe/Simferopol	65052
Europe/Skopje	65051	Europe/Sofia	65050
Europe/Stockholm	65049	Europe/Tallinn	65048
Europe/Tirane	65047	Europe/Tiraspol	65046
Europe/Ulyanovsk	65045	Europe/Uzhgorod	65044
Europe/Vaduz	65043	Europe/Vatican	65042
Europe/Vienna	65041	Europe/Vilnius	65040
Europe/Volgograd	65039	Europe/Warsaw	65038
Europe/Zagreb	65037	Europe/Zaporozhye	65036
Europe/Zurich	65035	Factory	65034
GB	65033	GB-Eire	65032
GMT+0	65031	GMT-0	65030
GMT0	65029	Greenwich	65028
HST	65027	Hongkong	65026

(разрыв таблицы)

(разрыв таблицы)

Часовой пояс	ID	Часовой пояс	ID
IET	65025	IST	65024
Iceland	65023	Indian/Antananarivo	65022
Indian/Chagos	65021	Indian/Christmas	65020
Indian/Cocos	65019	Indian/Comoro	65018
Indian/Kerguelen	65017	Indian/Mahe	65016
Indian/Maldives	65015	Indian/Mauritius	65014
Indian/Mayotte	65013	Indian/Reunion	65012
Iran	65011	Israel	65010
JST	65009	Jamaica	65008
Japan	65007	Kwajalein	65006
Libya	65005	MET	65004
MIT	65003	MST	65002
MST7MDT	65001	Mexico/BajaNorte	65000
Mexico/BajaSur	64999	Mexico/General	64998
NET	64997	NST	64996
NZ	64995	NZ-CHAT	64994
Navajo	64993	PLT	64992
PNT	64991	PRC	64990
PRT	64989	PST	64988
PST8PDT	64987	Pacific/Apia	64986
Pacific/Auckland	64985	Pacific/Bougainville	64984
Pacific/Chatham	64983	Pacific/Chuuk	64982
Pacific/Easter	64981	Pacific/Efate	64980
Pacific/Enderbury	64979	Pacific/Fakaofo	64978
Pacific/Fiji	64977	Pacific/Funafuti	64976
Pacific/Galapagos	64975	Pacific/Gambier	64974
Pacific/Guadalcanal	64973	Pacific/Guam	64972
Pacific/Honolulu	64971	Pacific/Johnston	64970
Pacific/Kiritimati	64969	Pacific/Kosrae	64968
Pacific/Kwajalein	64967	Pacific/Majuro	64966
Pacific/Marquesas	64965	Pacific/Midway	64964
Pacific/Nauru	64963	Pacific/Niue	64962
Pacific/Norfolk	64961	Pacific/Noumea	64960
Pacific/Pago_Pago	64959	Pacific/Palau	64958
Pacific/Pitcairn	64957	Pacific/Pohnpei	64956
Pacific/Ponape	64955	Pacific/Port_Moresby	64954
Pacific/Rarotonga	64953	Pacific/Saipan	64952

(разрыв таблицы)

(разрыв таблицы)

Часовой пояс	ID	Часовой пояс	ID
Pacific/Samoa	64951	Pacific/Tahiti	64950
Pacific/Tarawa	64949	Pacific/Tongatapu	64948
Pacific/Truk	64947	Pacific/Wake	64946
Pacific/Wallis	64945	Pacific/Yap	64944
Poland	64943	Portugal	64942
ROC	64941	ROK	64940
SST	64939	Singapore	64938
SystemV/AST4	64937	SystemV/AST4ADT	64936
SystemV/CST6	64935	SystemV/CST6CDT	64934
SystemV/EST5	64933	SystemV/EST5EDT	64932
SystemV/HST10	64931	SystemV/MST7	64930
SystemV/MST7MDT	64929	SystemV/PST8	64928
SystemV/PST8PDT	64927	SystemV/YST9	64926
SystemV/YST9YDT	64925	Turkey	64924
UCT	64923	US/Alaska	64922
US/Aleutian	64921	US/Arizona	64920
US/Central	64919	US/East-Indiana	64918
US/Eastern	64917	US/Hawaii	64916
US/Indiana-Starke	64915	US/Michigan	64914
US/Mountain	64913	US/Pacific	64912
US/Pacific-New	64911	US/Samoa	64910
UTC	64909	Universal	64908
VST	64907	W-SU	64906
WET	64905	Zulu	64904