# Ред База Данных

Версия 3.0

## Руководство по SQL

(Руководство пользователя)

46.29926343.502120-01 93 1

# Содержание

Введение 14

1 SQL Ред База Данных.
Общие сведения 16
1.1 Структура данных на внешнем носителе (ODS) . . . 16
1.2 Объекты базы данных . . . 17
1.3 Подмножества языка . . . 19
1.4 Диалекты базы данных . . . 19
1.5 Описание языковых конструкций . . . 20
1.6 Операторы. Общие сведения . . . 21
1.7 Ключевые слова. Общие сведения . . . 21
1.8 Идентификаторы . . . 22
1.9 Константы (литералы) . . . 22
Строковые константы . . . 23
Числовые константы . . . 24
Константы даты и времени . . . 25
1.10 Операторы SQL . . . 26
Оператор конкатенации . . . 27
Арифметические операторы . . . 27
Оператор сравнения . . . 27
Оператор равенства . . . 28
Логические операторы . . . 29
1.11 Комментарии . . . 30

2 Типы данных Ред База Данных 31
2.1 Перечень типов данных . . . 31
2.2 Предварительно определенные литералы и контекстные переменные . . . 33
2.3 Преобразование типов данных. Функция CAST . . . 36
Преобразование в целочисленные типы данных . . . 37
Преобразование в дробные числа с фиксированной точкой . . . 37
Преобразование в строковые типы данных . . . 37
Преобразование в типы данных даты и времени . . . 38
Преобразование в тип данных BLOB . . . 38
Преобразование в логический тип данных . . . 39
2.4 Числовые типы данных . . . 39
2.5 Строковые типы данных . . . 40
Функции для строковых типов данных . . . 40
2.6 Логический тип данных . . . 44
2.7 Типы данных даты и времени . . . 45
Тип данных DATE . . . 45
Тип данных TIME . . . 46
Тип данных TIMESTAMP . . . 47
Арифметические операции для типов данных даты и времени . . . 47
Функции для типов данных даты и времени . . . 48
2.8 Тип данных BLOB . . . 49
2.9 Тип данных SQL_NULL . . . 51

3 Работа с базой данных 52
3.1 Создание базы данных . . . 52
3.2 Примеры создания базы данных . . . 55
3.3 Соединение с существующей базой данных . . . 57
3.4 Изменение существующей базы данных . . . 59
3.5 Удаление базы данных . . . 61
3.6 Создание примечаний для базы данных и объектов базы данных . . . 61
3.7 Использование оперативных копий . . . 62
Создание оперативной копии . . . 63
Удаление оперативной копии . . . 65

4 Работа с доменами 66
4.1 Создание домена . . . 66
Задание типа данных . . . 66
Значение по умолчанию . . . 67
Значение NOT NULL . . . 67
Синтаксис ограничения домена . . . 67
Предикаты сравнения . . . 73
Логические операции с условиями домена . . . 78
4.2 Изменение домена . . . 80
4.3 Удаление домена . . . 81
4.4 Примечание домена . . . 82

5 Работа с таблицами 83
5.1 Создание таблиц . . . 83
Глобальные временные таблицы (GTTs) . . . 83
Использование внешних файлов . . . 84
Файлы с фиксированной длиной строк . . . 85
Файлы формата CSV . . . 86
Определение столбца . . . 87
Синтаксис определения столбца . . . 87
Задание типа данных . . . 88
Использование ссылки на домен . . . 88
Значение по умолчанию . . . 89
Значение NOT NULL . . . 89
Ограничение столбца . . . 90
Вычисляемые столбцы . . . 101
Столбцы идентификации . . . 104
Определение ограничений таблицы . . . 104
Ограничение первичного ключа . . . 105
Ограничение уникального ключа . . . 105
Ограничение внешнего ключа . . . 106
Ограничение CHECK . . . 107
5.2 Изменение таблиц . . . 107
Добавление нового столбца . . . 108
Добавление ограничения таблицы . . . 109
Удаление столбца таблицы . . . 110
Удаление ограничения . . . 111
Изменение существующего столбца . . . 111
Изменение имени . . . 112
Изменение типа данных . . . 112
Изменение позиции столбца . . . 112
Удаление значения по умолчанию столбца . . . 113
Добавление значения по умолчанию столбца . . . 113
Добавление и удаление ограничения NOT NULL . . . 113

Изменение типа и выражения для вычисляемого столбца . . . 113
Изменение столбцов идентификации . . . 113
Изменение прав на работу с таблицей . . . 114
5.3 Удаление таблиц . . . 114
5.4 Пересоздание таблицы . . . 114
5.5 Примечание к таблице и ее столбцам . . . 114

6 Работа с генераторами 116
6.1 Создание генератора . . . 116
6.2 Изменение значения генератора . . . 116
6.3 Создание нового или изменение существующего генератора . . . 117
6.4 Удаление генератора . . . 118
6.5 Пересоздание генератора . . . 118
6.6 Примечание к генератору . . . 118

7 Работа с индексами 119
7.1 Создание индекса . . . 119
Ограничения на индексы . . . 120
7.2 Изменение индекса . . . 121
7.3 Удаление индекса . . . 121
7.4 Селективность индекса . . . 122
7.5 Примечание индекса . . . 122

8 Операторы DML 124
8.1 SELECT . . . 124
Предложение WITH RECURSIVE . . . 125
Список выбора . . . 128
Использование ключевых слов FIRST и SKIP . . . 129
Предложение FROM . . . 130
Выборка из таблицы или представления . . . 132
Выборка из селективной хранимой процедуры . . . 132
Выборка из производной таблицы . . . 132
Выборка из общих табличных выражений . . . 133
Соединение таблиц (JOIN) . . . 134
Внутреннее соединение . . . 134
Внешние соединения . . . 138
Перекрестное соединение . . . 141
Соединения именованными столбцами . . . 141
Естественное соединение . . . 142
Предложение WHERE . . . 143
Синтаксис условия выборки данных . . . 143
Оператор сравнения . . . 143
Значение в условии выборки данных . . . 144
Использование встроенных функций . . . 145
Операторы SELECT, используемые в условии выборки данных . . . 146
Оператор IN . . . 146
Оператор BETWEEN . . . 146
Оператор LIKE . . . 147
Оператор IS NULL . . . 147
Оператор IS DISTINCT FROM . . . 148
Функции ALL, SOME, ANY . . . 148
Функция EXISTS . . . 148
Функция SINGULAR . . . 148
Оператор CONTAINING . . . 148
Оператор STARTING WITH . . . 149

Логические операции с условиями выборки . . . 149
Предложения GROUP BY и HAVING . . . 150
Предложение UNION . . . 152
Предложение PLAN . . . 156
Примеры простых планов . . . 157
Примеры составных планов . . . 160
Предложение ORDER BY . . . 165
Предложение OPTIMIZE FOR . . . 166
Предложение ROWS . . . 167
Соотношения между ключевыми словами FIRST и SKIP и предложением ROWS 167
Предложение FETCH, OFFSET . . . 168
Предложение FOR UPDATE . . . 168
Предложение WITH LOCK . . . 168
Предложение INTO . . . 170
8.2 INSERT . . . 170
Использование ключевого слова VALUES . . . 171
Использование поиска многих . . . 173
Использование DEFAULT VALUES . . . 174
Предложение RETURNING . . . 174
8.3 UPDATE . . . 174
Предложение SET . . . 175
Предложение WHERE . . . 176
Предложение PLAN . . . 176
Предложение ORDER BY и ROWS . . . 176
Предложение RETURNING . . . 177
Примеры . . . 177
8.4 UPDATE OR INSERT . . . 178
8.5 DELETE . . . 179
Примеры . . . 180
8.6 EXECUTE BLOCK . . . 181

9 Работа с представлениями 184
9.1 Создание представлений . . . 184
9.2 Изменение представлений . . . 185
9.3 Создание или изменение представлений . . . 186
9.4 Удаление представлений . . . 186
9.5 Пересоздание представлений . . . 186
9.6 Примеры представлений . . . 187
9.7 Преобразование неизменяемых представлений в изменяемые при помощи триггеров . 190
9.8 Системные представления . . . 194
9.9 Примечание представления . . . 196

10 Транзакции 197
10.1 Старт транзакции . . . 198
Средства резервирования . . . 199
Режим доступа . . . 200
Режим разрешения блокировок . . . 200
Уровень изоляции . . . 201
Уровень изоляции SNAPSHOT . . . 201
Уровень изоляции SNAPSHOT TABLE STABILITY . . . 202
Уровень изоляции READ COMMITTED . . . 203
Опция NO AUTO UNDO . . . 204
Опция IGNORE LIMBO . . . 205
10.2 Подтверждение транзакции . . . 205
10.3 Откат (отмена) транзакции . . . 205

10.4 Использование вложенных транзакций . . . 206
10.5 Вариант взаимной блокировки . . . 207

11 Хранимые процедуры, хранимые функции, триггеры и пакеты 209
11.1 Язык хранимых процедур, функций и триггеров . . . 209
Использование оператора SET TERM . . . 210
Внутренние переменные . . . 210
Предварительно определенные литералы и контекстные переменные . . . 212
11.2 Пользовательские исключения . . . 212
11.3 События базы данных . . . 214
11.4 Операторы языка хранимых процедур, функций и триггеров . . . 215
Объявление локальных переменных и курсоров . . . 215
Объявление подпроцедуры . . . 216
Объявление подфункции . . . 217
Операция присваивания . . . 217
Оператор IF-THEN-ELSE . . . 219
Оператор WHILE-DO . . . 219
Операторы перехода . . . 220
Оператор EXIT . . . 220
Оператор LEAVE . . . 220
Оператор SUSPEND . . . 221
Оператор CONTINUE . . . 221
Оператор EXECUTE PROCEDURE . . . 222
Обычные операторы обращения к базе данных . . . 222
Оператор FOR SELECT-DO . . . 223
Оператор FOR EXECUTE STATEMENT . . . 224
Использование курсоров . . . 225
Оператор IN AUTONOMOUS TRANSACTION . . . 227
Обработка ошибочных ситуаций . . . 228
11.5 Работа с триггерами . . . 229
DML триггеры . . . 229
Триггеры на события базы данных . . . 230
DDL триггеры . . . 230
Создание триггера . . . 231
Изменение триггера . . . 234
Создание нового или изменение существующего триггера . . . 234
Удаление триггера . . . 235
Создание нового или пересоздание существующего триггера . . . 235
Примеры триггеров . . . 236
Формирование значения искусственного первичного ключа . . . 236
Передача сообщений клиентским процессам об изменении данных . . . 236
Пример триггера, обеспечивающего поддержание ссылочной целостности данных 237
Пример триггера, создающего запись истории окладов . . . 238
Триггеры, преобразующие неизменяемые представления в изменяемые . . . 239
11.6 Работа с хранимыми процедурами . . . 239
Создание хранимой процедуры . . . 239
Изменение хранимой процедуры . . . 241
Создание новой или изменение существующей хранимой процедуры . . . 242
Удаление хранимой процедуры . . . 242
Создание новой или пересоздание существующей хранимой процедуры . . . 243
Примеры хранимых процедур . . . 243
Получение значения искусственного первичного ключа . . . 243
Вычисление факториала числа . . . 244

11.7 Работа с хранимыми функциями . . . 249
Создание хранимой функции . . . 249
Изменение хранимой функции . . . 250
Создание новой или изменение существующей хранимой функции . . . 251
Удаление хранимой функции . . . 251
Создание новой или пересоздание существующей хранимой функции . . . 252
11.8 Работа с пакетами . . . 252
Создание заголовка пакета . . . 253
Изменение заголовка пакета . . . 254
Создание нового или изменение существующего заголовка пакета . . . 254
Удаление заголовка пакета . . . 255
Создание нового или пересоздание существующего заголовка объекта . . . 255
Создание тела пакета . . . 256
Удаление тела пакета . . . 257
Создание нового или пересоздание существующего тела объекта . . . 257

12 Внешние хранимые процедуры, функции и триггеры, написанные на языке Java 259
12.1 Объявление/изменение/пересоздание внешних процедур . . . 259
Примеры . . . 259
12.2 Объявление/изменение/пересоздание внешних функций . . . 260
Примеры . . . 261
12.3 Объявление/изменение/пересоздание внешних триггеров . . . 261
Примеры . . . 262
12.4 Удаление внешних процедур, функций и триггеров . . . 262
12.5 Вызов внешних процедур и функций . . . 263

13 Полнотекстовый поиск 264
13.1 Настройка сервера для работы полнотекстового поиска . . . 264
Безопасность . . . 267
13.2 Служебные объекты для полнотекстового поиска . . . 269
Служебные домены . . . 269
Служебные таблицы . . . 269
FTS$INDICES . . . 269
FTS$INDEX_SEGMENTS . . . 270
FTS$POOL . . . 270
Служебные хранимые процедуры . . . 271
FTS$CREATE_INDEX . . . 271
FTS$DROP_INDEX . . . 272
FTS$ADD_FIELD_TO_INDEX . . . 272
FTS$DROP_FIELD_FROM_INDEX . . . 272
FTS$REINDEX . . . 273
FTS$SEARCH . . . 273
FTS$FULL_REINDEX . . . 273
FTS$STARTDAEMON . . . 274
FTS$STOPDAEMON . . . 274
13.3 Пример использования полнотекстового поиска . . . 274
Создание индекса . . . 274
Добавление полей в индекс . . . 275
Удаление полей из индекса . . . 275
Переиндексация . . . 275
Поиск . . . 275
Удаление индекса . . . 276
13.4 Синтаксис поисковых запросов . . . 277
Термы . . . 277
Маска . . . 277

Нечеткий поиск . . . 277
Усиление термов . . . 277
Логические операции . . . 277
Экранирование специальных символов . . . 279
Приложение А Зарезервированные и ключевые слова . . . 280
Приложение Б Коды ошибок Ред База Данных . . . 285
Б.1 Коды ошибок GDSCODE и SQLCODE . . . 285
Б.2 Коды ошибок SQLSTATE . . . 370
Приложение В Наборы символов и порядки сортировки . . . 378
Приложение Г Функции, определенные пользователем (UDF) . . . 383
Объявление в базе данных UDF . . . 383
Изменение точки входа или имени модуля для UDF . . . 384
Удаление объявления UDF из базы данных . . . 384
Приложение Д Операторы SQL . . . 386
ALTER CHARACTER SET . . . 386
ALTER DATABASE . . . 386
ALTER DOMAIN . . . 388
ALTER EXCEPTION . . . 388
ALTER EXTERNAL FUNCTION . . . 389
ALTER FUNCTION . . . 389
ALTER INDEX . . . 390
ALTER MAPPING . . . 390
ALTER PACKAGE . . . 391
ALTER POLICY . . . 391
ALTER PROCEDURE . . . 392
ALTER ROLE . . . 392
ALTER SEQUENCE . . . 393
ALTER TABLE . . . 393
ALTER TRIGGER . . . 395
ALTER USER . . . 396
ALTER VIEW . . . 397
COMMENT . . . 398
COMMIT . . . 398
CONNECT . . . 399
CREATE COLLATION . . . 399
CREATE DATABASE . . . 401
CREATE DOMAIN . . . 402
CREATE EXCEPTION . . . 406
CREATE FUNCTION . . . 406
CREATE GENERATOR . . . 408
CREATE GLOBAL TEMPORARY TABLE . . . 408
CREATE INDEX . . . 409
CREATE MAPPING . . . 409
CREATE OR ALTER EXCEPTION . . . 410
CREATE OR ALTER FUNCTION . . . 410
CREATE OR ALTER MAPPING . . . 411
CREATE OR ALTER PACKAGE . . . 411
CREATE OR ALTER PROCEDURE . . . 412
CREATE OR ALTER SEQUENCE . . . 413
CREATE OR ALTER TRIGGER . . . 413
CREATE OR ALTER USER . . . 415
CREATE OR ALTER VIEW . . . 415
CREATE PACKAGE . . . 416
CREATE PACKAGE BODY . . . 417

CREATE POLICY . . . 418
CREATE PROCEDURE . . . 419
CREATE ROLE . . . 421
CREATE SEQUENCE . . . 421
CREATE SHADOW . . . 421
CREATE TABLE . . . 422
Определение столбца . . . 423
Ограничение таблицы . . . 428
CREATE TRIGGER . . . 429
CREATE USER . . . 432
CREATE VIEW . . . 433
DECLARE EXTERNAL FUNCTION . . . 434
DECLARE FILTER . . . 435
DELETE . . . 436
DROP COLLATION . . . 437
DROP DATABASE . . . 438
DROP DOMAIN . . . 438
DROP EXCEPTION . . . 438
DROP EXTERNAL FUNCTION . . . 438
DROP FILTER . . . 439
DROP GENERATOR . . . 439
DROP INDEX . . . 439
DROP FUNCTION . . . 440
DROP MAPPING . . . 440
DROP PACKAGE . . . 440
DROP PACKAGE BODY . . . 440
DROP POLICY . . . 441
DROP PROCEDURE . . . 441
DROP ROLE . . . 441
DROP SEQUENCE . . . 441
DROP SHADOW . . . 442
DROP TABLE . . . 442
DROP TRIGGER . . . 442
DROP USER . . . 443
DROP VIEW . . . 443
EXECUTE BLOCK . . . 443
EXECUTE PROCEDURE . . . 444
GRANT . . . 444
Привилегии для таблиц и представлений . . . 446
Привилегии EXECUTE . . . 447
Привилегии USAGE . . . 447
Привилегии на выполнение DDL операций . . . 447
DDL привилегии на базу данных . . . 447
Назначение ролей . . . 448
Назначение политик . . . 448
INSERT . . . 449
MERGE . . . 450
RECREATE EXCEPTION . . . 452
RECREATE FUNCTION . . . 452
RECREATE PACKAGE . . . 452
RECREATE PACKAGE BODY . . . 453
RECREATE PROCEDURE . . . 454
RECREATE SEQUENCE . . . 455
RECREATE TABLE . . . 456

RECREATE TRIGGER . . . 456
RECREATE VIEW . . . 457
RELEASE SAVEPOINT . . . 458
RESET USER . . . 458
REVOKE . . . 458
ROLLBACK . . . 461
SAVEPOINT . . . 461
SELECT . . . 462
Список выбора . . . 463
Предложение FROM . . . 464
Предложение WHERE . . . 465
Предложение GROUP BY и HAVING . . . 467
Предложение UNION . . . 468
Предложение PLAN . . . 468
Предложение ORDER BY, ROWS, OFFSET, FETCH . . . 469
Предложение FOR UPDATE, WITH LOCK и INTO . . . 470
Предложение OPTIMIZE FOR . . . 470
SET GENERATOR . . . 470
SET NAMES . . . 471
SET ROLE . . . 471
SET SQL DIALECT . . . 471
SET STATISTICS . . . 472
SET TRANSACTION . . . 472
SET TRUSTED ROLE . . . 475
SIMILAR TO . . . 476
UPDATE . . . 479
UPDATE OR INSERT . . . 481
Приложение E Функции . . . 483
ABS() . . . 483
ACOS() . . . 483
ACOSH() . . . 483
ASCII_CHAR() . . . 484
ASCII_VAL() . . . 484
ASIN() . . . 484
ASINH() . . . 484
ATAN() . . . 485
ATAN2() . . . 485
ATANH() . . . 485
AVG() . . . 486
BIN_AND() . . . 486
BIN_NOT() . . . 486
BIN_OR() . . . 487
BIN_SHL() . . . 487
BIN_SHR() . . . 487
BIN_XOR() . . . 488
BIT_LENGTH() . . . 488
CASE-WHEN-ELSE() . . . 488
CAST() . . . 489
CEIL|CEILING() . . . 490
CHAR_TO_UUID() . . . 490
CHARACTER_LENGTH() . . . 490
CHECK_DDL_RIGHTS() . . . 491
CHECK_DML_RIGHTS() . . . 491
COALESCE() . . . 491

CORR() . . . 492
COS() . . . 492
COSH() . . . 492
COT() . . . 493
COUNT() . . . 493
COVAR_POP() . . . 493
COVAR_SAMP() . . . 494
CPU_LOAD() . . . 494
CREATE_FILE() . . . 494
DAMLEV() . . . 495
DATEADD() . . . 495
DATEDIFF() . . . 496
DECODE() . . . 496
DELETE_FILE() . . . 497
EXP() . . . 497
EXTRACT() . . . 497
FLOOR() . . . 498
GEN_ID() . . . 498
GEN_UUID() . . . 499
HASH() . . . 499
BLOB_APPEND . . . 499
HASH_CP() . . . 501
HASHAGG() . . . 501
IIF() . . . 501
LDAP_ATTR() . . . 501
LEFT() . . . 502
LIST() . . . 502
LN() . . . 502
LOG() . . . 503
LOG10() . . . 503
LOWER() . . . 503
LPAD() . . . 503
MAX() . . . 504
MAXVALUE() . . . 505
MIN() . . . 505
MINVALUE() . . . 505
MOD() . . . 505
NEXT VALUE FOR . . . 506
NULLIF() . . . 506
OCTET_LENGTH() . . . 506
OVER() . . . 506
- Агрегатные функции . . . 507
- Секционирование . . . 507
- Сортировка . . . 507
- Ранжирующие функции . . . 507
- Навигационные функции . . . 508
- Агрегатные функции внутри оконных . . . 508

OVERLAY() . . . 509
PI() . . . 509
POSITION() . . . 509
POWER() . . . 509
RAND() . . . 510
RDB$GET_CONTEXT() . . . 510
RDB$SET_CONTEXT() . . . 513

READ_FILE() . . . 513
REGEXP_SUBSTR() . . . 513
REGR_AVGX() . . . 515
REGR_AVGY() . . . 515
REGR_COUNT() . . . 516
REGR_INTERCEPT() . . . 516
REGR_R2() . . . 516
REGR_SLOPE() . . . 517
REGR_SXX() . . . 517
REGR_SXY() . . . 518
REGR_SYY() . . . 518
REPLACE() . . . 519
REVERSE() . . . 519
RIGHT() . . . 519
ROUND() . . . 519
RPAD() . . . 520
SIGN() . . . 520
SIN() . . . 521
SINH() . . . 521
SQRT() . . . 521
STDDEV_POP() . . . 521
STDDEV_SAMP() . . . 522
SUBSTRING() . . . 522
SUM() . . . 523
TAN() . . . 523
TANH() . . . 523
TRIM() . . . 524
TRUNC() . . . 524
UPPER() . . . 524
UTC_TIMESTAMP() . . . 525
UUID_TO_CHAR() . . . 525
VAR_POP() . . . 525
VAR_SAMP() . . . 526
Приложение Ж Контекстные переменные . . . 527
CURRENT_CONNECTION . . . 527
CURRENT_DATE . . . 527
CURRENT_ROLE . . . 527
CURRENT_TIME . . . 527
CURRENT_TIMESTAMP . . . 527
CURRENT_TRANSACTION . . . 528
CURRENT_USER . . . 528
INSERTING, UPDATING и DELETING . . . 528
NEW . . . 528
NOW . . . 528
OLD . . . 529
ROW_COUNT . . . 529
SQLCODE, GDSCODE . . . 529
SQLSTATE . . . 529
TODAY . . . 530
TOMORROW . . . 530
USER . . . 530
YESTERDAY . . . 530
Приложение З Операторы языка хранимых процедур, функций и триггеров . . . 531
CLOSE . . . 531

CONTINUE . . . . . 531
DECLARE CURSOR . . . . . 531
DECLARE FUNCTION . . . . . 532
DECLARE PROCEDURE . . . . . 532
DECLARE VARIABLE . . . . . 533
EXCEPTION . . . . . 534
EXIT . . . . . 534
FETCH . . . . . 535
FOR EXECUTE STATEMENT . . . . . 536
Предложение ON EXTERNAL [DATA SOURCE] . . . . . 538
FOR SELECT-DO . . . . . 540
IF-THEN-ELSE . . . . . 540
LEAVE . . . . . 541
OPEN . . . . . 541
POST_EVENT . . . . . 541
SUSPEND . . . . . 541
WHEN-DO . . . . . 542
WHILE-DO . . . . . 542

# 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 12 глав и 8 приложений.

**Глава 1 «SQL Ред База Данных. Общие сведения»** 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.

**Глава 2 «Типы данных Ред База Данных»** 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.

**В главе 3 «Работа с базой данных»** описываются языковые средства по созданию, изменению и удалению баз данных, средства создания и удаления оперативных копий (shadow), подключение к существующей базе данных, удаление базы данных, приводится множество примеров создания однофайловых и многофайловых баз данных.

**В главе 4 «Работа с доменами»** подробно рассматриваются языковые средства создания, изменения и удаления доменов.

**В главе 5 «Работа с таблицами»** 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.

**В главе 6 «Работа с генераторами»** описываются средства работы с генераторами — создание, удаление, изменение текущего значения генераторов.

**В главе 7 «Работа с индексами»** описывается порядок использования индексов для таблиц. Рассматриваются операторы создания индекса, изменения активности, удаления и улучшения селективности (избирательности) индекса.

**Глава 8 «Операторы DML»** 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 9 «Работа с представлениями»** 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.

**Глава 10 «Транзакции»** содержит детальное описание характеристик транзакций — режимов доступа, уровней изоляции, режимов разрешения блокировок, средств резервирования. Приводится синтаксис операторов работы с транзакциями.

**В главе 11 «Хранимые процедуры, хранимые функции, триггеры и пакеты»** описывается язык хранимых процедур, функций и триггеров PSQL. Приводятся примеры различных

выполняемых хранимых процедур, хранимых процедур выбора и триггеров.

**В главе 12 «Внешние хранимые процедуры, функции и триггеры, написанные на языке Java»** описывается синтаксис операторов создания, изменения и удаления внешних хранимых процедур, функций и триггеров, написанных на Java. Приводятся примеры.

**В приложении А «Зарезервированные и ключевые слова»** 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).

**Приложение Б «Коды ошибок Ред База Данных»** содержит список кодов ошибок, которые могут возникнуть при работе с базой данных.

**В приложении В «Наборы символов и порядки сортировки»** приводится список доступных наборов символов. Для каждого набора символов перечисляются порядки сортировки.

**В приложении Г «Функции, определенные пользователем (UDF)»** описываются функции, определенные пользователем, которые поставляются вместе с системой. Задается форма обращения к этим функциям.

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

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

**Приложение Ж «Контекстные переменные»** 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.

**Приложение З «Операторы языка хранимых процедур, функций и триггеров»** 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 1

# SQL Ред База 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 (иногда произносится «эс-кью-эль» или, чаще, «сиквел») — 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 База Данных имеет страничную довольно сложную организацию.

В разных версиях Ред База 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 База Данных используется ODS - RD_1. (Последняя версия Firebird 2.1 использует ODS 11.2). 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 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 База Данных. Однако при таком копировании/восстановлении базы данных все равно нет точных гарантий, что преобразование ODS будет выполнено правильно. Более надежным способом является выполнение для старой версии базы данных выделение данных

(`extract` — вывод метаданных и данных всех таблиц в виде операторов `CREATE, ALTER` и `INSERT`), создание новой базы данных в новой версии ODS и выполнение полученных операторов.

## 1.2 Объекты базы данных

Объектами базы данных в СУБД Ред База Данных являются следующие.

**Таблица** (`table`). Это основной объектлюбой реляционной базы данных, в том числе, и Ред База 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 может содержать не менее одного столбца. Каждый столбец имеет конкретный тип данных (`datatype`). В базах данных используется некоторое количество типов данных, похожие на те, которые применяются в обычных языках программирования. Подробнее о типах данных Ред База Данных см. в главе 2 «Типы данных Ред База Данных». О создании и изменении таблиц см. в главе 5 «Работа с таблицами».

**Домен** (`domain`) — объектбазы 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 4 «Работа с доменами».

**Индекс** (`index`) — объектбазы 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 7 «Работа с индексами».

**Генератор** (`generator`, **или** `sequence`) — простой объектреляционной 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 6 «Работа с генераторами».

**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 11 «Хранимые процедуры, хранимые функции, триггеры и пакеты». В Ред База Данных также существуют дополнительные расширения языка хранимых процедур с добавлением элементов языка Java. Подробности см. в документе «Руководство по расширенным хранимым процедурам».

**Хранимая функция** (`stored function`) — программа, написанная на процедурном расширении языка SQL (который также называется языком 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 11 «Хранимые процедуры, хранимые функции, триггеры и пакеты».

**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 База 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 11 «Хранимые процедуры, хранимые функции, триггеры и пакеты».

**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 11 «Хранимые процедуры, хранимые функции, триггеры и пакеты».

**Пользовательские исключения** (`exception`). Объектреляционной 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 11 «Хранимые процедуры, хранимые функции, триггеры и пакеты».

**События базы данных** (`event`). В Ред База 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 11 «Хранимые процедуры, хранимые функции, триггеры и пакеты».

**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 9 «Работа с представлениями».

**Функции, определенные пользователем** (`User Defined Functions, UDF`) — 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 База Данных представлено достаточно большое количество полезных функций, созданных разработчиками системы. Подробности описания в базе данных и использования `UDF` см. в приложении Г «Функции, определенные пользователем (`UDF`)».

**Транзакция** (`transaction`). Это не объектбазы данных, а некоторый механизм, используе-

мый 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 База 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 10 «Транзакции».

## 1.3 Подмножества языка

В SQL СУБД Ред База 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 11 «Хранимые процедуры, хранимые функции, триггеры и пакеты».

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, использования 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 База 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 Описание языковых конструкций

Для описания синтаксиса базовых элементов и других более сложных конструкций SQL используются нотации(система обозначений) Бэкуса-Наура, чаще всего применяемые для описания синтаксиса любых достаточно сложных формальных языков, в том числе, и языков программирования. Обычно используется несколько расширенный вариант этих нотаций. В таком варианте, как минимум в виде дополнения к принятой классической системе обозначений, используется символ многоточия (три подряд идущие точки — ...) для того, чтобы указать, что предыдущая конструкция (нетерминальный символ или любая, сколь угодно сложная конструкция, заключенная в фигурные

или квадратные скобки) может повторяться произвольное количество раз. В основе любого формального языка, которым является и SQL, лежат: алфавит языка (терминальные, основные, символы), более сложные конструкции, такие как имена (идентификаторы), литералы, конструкции еще более высокого уровня — операторы, предложения и др. Нетерминальными символами являются любые слова, заключенные в символы < и >. Нетерминальные символы должны быть уточнены в дальнейших описаниях синтаксиса языка при помощи терминальных символов. Семантика, смысл, языковых конструкций описывается обычным естественным языком.

Алфавит SQL состоит из букв, цифр и специальных символов.

```
<буква> ::= <прописаная 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
```

Буквой в SQL является прописная или строчная буква только латинского алфавита. Буквы кириллицы не относятся к категории букв SQL. Различий между прописными и строчными буквами в этом языке не существует за исключением использования букв в именах с разделителями (см. ниже). Цифрой в синтаксисе SQL является обычная десятичная цифра в диапазоне от 0 до 9:

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

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.

## 1.6 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.).

## 1.7 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 приложении А.

## 1.8 Идентификаторы

Все объекты базы данных имеют имена, которые иногда называют идентификаторами. 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.

Следующие имена с точки зрения системы являются одинаковыми:

```
fullname        FULLNAME        FuLlNaMe        FullName
```

В нотациях Бэкуса-Наура имя (идентификатор) определяется следующим образом:

**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
```

В имени нельзя использовать буквы кириллицы, пробелы, другие специальные символы.

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.

Синтаксис имени с разделителями:

**Листинг 1.2. Синтаксис имен с разделителями**

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

Многоточие в синтаксисе означает, что предыдущая конструкция (в нашем случае «любой символ ASCII») может повторяться произвольное количество раз.

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.

## 1.9 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` |

## 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}'<шестнадцатиричная строка>'

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

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

```
SELECT x'4E657276656E' FROM rdb$database
-- returns 4E657276656E, a 6-byte 'binary' string
SELECT _ascii x'4E657276656E' FROM rdb$database
-- returns 'Nerven' (same string, now interpreted as ASCII text)
SELECT _iso8859_1 x'53E46765' FROM rdb$database
-- returns 'Säge' (4 chars, 4 bytes)
SELECT _utf8 x'53C3A46765' FROM rdb$database
-- returns '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.

## 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. Включение 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.
- Шестнадцатеричная запись чисел.

Целочисленные значения могут быть записаны в шестнадцатеричной системе счисления. Числа состоящие из 1-8 шестнадцатеричных цифр будут интерпретированы как `INTEGER`, состоящие из 9-16 цифр — как `BIGINT`.

```
{x|X} <шестнадцатиричные число>

<шестнадцатиричные число> ::= 1-16 <шестнадцатиричных цифр>

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

```
SELECT 0x4F9 FROM rdb$database -- returns 1273
SELECT 0xFFFFFFFFFFFFFFFF FROM rdb$database -- returns -1
SELECT 0x9E44F9A8 FROM rdb$database -- returns -1639646808 (INTEGER)
SELECT 0x09E44F9A8 FROM rdb$database -- returns 2655320488 (BIGINT)
```

- Шестнадцатеричные числа в диапазоне `0 .. 7FFF FFFF` являются положительными `INTEGER` числа со значениями `0 .. 2147483647`. Для того, чтобы интерпретировать константу как `BIGINT` число необходимо дописать необходимо количества нулей слева. Это изменит тип, но не значение.
- Числа в диапазоне `8000 0000 .. FFFF FFFF`:
  - При записи восемью шестнадцатеричный числами, такие как `0x9E44F9A8`, интерпретируется как 32-битное целое. Поскольку крайний левый (знаковый) бит установлен, то такие числа будут находится в отрицательном диапазоне `-2147483648 .. -1`.
  - Числа предварённые одним или несколькими нулями, такие как `0x09E44F9A8`, будут интерпретированы как 64-разрядный `BIGINT` в диапазоне значений `0000 0000 8000 0000 .. 0000 0000 FFFF FFFF`. В этом случае знаковый бит не установлен, поэтому они отображаются в положительном диапазоне `2147483648 .. 4294967295` десятичных чисел.
    Таким образом, только в этом диапазоне числа, предварённые совершенно незначимым нулём, имеют кардинально разные значения. Это необходимо знать.
- Шестнадцатеричные числа в диапазоне `1 0000 0000 .. 7FFF FFFF FFFF FFFF` являются положительными `BIGINT` числами.
- Шестнадцатеричные числа в диапазоне `8000 0000 0000 0000 .. FFFF FFFF FFFF FFFF` являются отрицательными `BIGINT` числами.
- 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. Вы можете записать отрицательное `SMALLINT` число в шестнадцатеричном виде используя 4-байтное шестнадцатеричное число в диапазоне от `0xFFFF8000` (десятичное −32768) до `0xFFFFFFFF` (десятичное -1).

## 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 2 «Типы данных Ред База Данных».

## 1.10 Операторы SQL

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

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

<строковый оператор> ::= ||

<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. |

| Тип оператора | Приоритет | Пояснение |
|---|---|---|
| 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.

## 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
```

## Арифметические операторы

Таблица 1.5 — Приоритет арифметических операторов

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

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

## 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 8.1

Таблица 1.6 — Приоритет операторов сравнения

| Оператор | Назначение | 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
```

## 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 | Назначение | 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 |
| **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 конъюнкции 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 |

## 1.11 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.

```
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
```

Глава 2

# Типы данных Ред База Данных

## 2.1 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 2.1. 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 5 «Работа с таблицами» и в главе 11 «Хранимые процедуры, хранимые функции, триггеры и пакеты»).

Таблица 2.1 — Типы данных, используемые в Ред База 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`. |
| `BLOB` | Переменный | Большой двоичный объект (Binary Large OBject). Позволяет хранить произвольные данные: форматированные тексты, графику, звуки, видеофильмы. |
| `CHAR(n)`<br>`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 В).<br>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. |

| Тип данных | Размер (байт) | Описание |
|---|---|---|
| `DECIMAL(n, m)` | 2, 4 или 8 | 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 18), `m` — количество знаков после десятичной точки (значения от 0 до 18). Основное требование к параметрам: `m <= n`.<br>Если значения для `n` и/или `m` не указаны, то предполагается (9, 0).<br>Для этого типа данных допустимо сокращение `DEC`. |
| `DOUBLE PRECISION` | 8 | Числовой тип данных. Число с плавающей точкой. Находится в диапазоне от $2.225 \times 10^{-308}$ до $1.179 \times 10^{308}$. Позволяет хранить до 15 значащих цифр. |
| `FLOAT` | 4 | Числовой тип данных. Число с плавающей точкой. Диапазон хранимых чисел от $1.175 \times 10^{-38}$ до $3.402 \times 10^{38}$. Позволяет сохранять до 7 значащих цифр. |
| `INTEGER`<br>`INT` | 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.<br>Для этого типа данных допустимо сокращение `INT`. |
| `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.<br>Если количество символов `n` не указано, принимается 1.<br>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.<br>Количество символов `n` обязательно должно быть указано.<br>Допустимы следующие сокращения: `NATIONAL CHAR VARYING, NCHAR VARYING`. |
| `NUMERIC(n, m)` | 2, 4 или 8 | Числовой тип данных. Дробное или целое число с фиксированной точкой, `n` — обще количество знаков в числе от 1 до 18, `m` — количество знаков после десятичной точки (число от 0 до 18). Общее требование: `m <= n`.<br>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. |
| `TIME` | 4 | Задает время в часах, минутах, секундах и десятитысячных долях секунды. Диапазон: от 00:00:00.0000 до 23:59:59.9999. |
| `TIMESTAMP` | 8 | Комбинация (объединение) даты и времени. |

| Тип данных | Размер (байт) | Описание |
|---|---|---|
| `VARYING CHARACTER (n)` | n символов | Символьный тип данных переменной длины. Максимальный размер 32765 байтов. Количество хранимых символов зависит от используемого для столбца набора символов (см. приложение B, где для каждого набора символов указано и количество байтов, необходимых для хранения одного символа).<br>Для этого типа данных, в отличие от `CHAR` (где по умолчанию предполагается количество символов 1), количество символов `n` обязательно должно быть указано.<br>Допустимо сокращение `VARCHAR`. |

Типы данных можно объединить в группы, или категории — числовые данные, строковые, данные даты и времени, а также единственный в группе двоичный тип данных `BLOB`. Его можно промоделировать при помощи типа данных `CHAR(1)`. Такой пример, точнее, два примера, будут рассмотрены несколько позже в этой главе.

Любой тип данных, кроме `BLOB`, может быть также массивом произвольной размерности.

## 2.2 Предварительно определенные литералы и контекстные переменные

### 'NOW', 'TODAY', 'TOMORROW', 'YESTERDAY'

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. В настоящей версии системы существует четыре предварительно определенных литерала даты.

- `'NOW'` типа `TIMESTAMP` возвращает текущую дату и текущее время на сервере, включая миллисекунды. При обращении к этому литералу возвращаются не четыре знака после десятичной точки, как в типах данных `TIME` и `TIMESTAMP`, а только три;
- `'TODAY'` типа `DATE` возвращает текущую дату сервера;
- `'TOMORROW'` типа `DATE` возвращает завтрашнюю дату сервера;
- `'YESTERDAY'` типа `DATE` возвращает вчерашнюю дату сервера.

Эти литералы не являются чувствительными к регистру — их можно вводить как прописными буквами, так и строчными. Их также можно записывать, используя прописные и строчные буквы вперемешку. При использовании этих литералов в операторе `SELECT` и в некоторых других операторах обращения к базе данных необходимо явно выполнить преобразование к соответствующему типу данных с использованием функции `CAST`. Иначе во многих случаях они будут рассматриваться просто как строковые константы.

### CURRENT_TIMESTAMP

Контекстная переменная `CURRENT_TIMESTAMP` типа `TIMESTAMP` возвращает текущую дату и текущее время сервера. Во времени могут указываться и миллисекунды — три знака после десятичной точки при указании количества секунд. Возвращает такое же значение, что и предварительно определенный литерал `'NOW'`. При обращении к этой контекстной переменной можно задавать в виде параметра 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 (в отличие от CURRENT_TIME, где по умолчанию принимается 0).

### CURRENT_DATE и CURRENT_TIME

Вместо обращения к контекстной переменной CURRENT_TIMESTAMP для получения того же значения можно использовать следующее выражение, содержащее две контекстные переменные CURRENT_DATE и CURRENT_TIME:

```
CAST(CURRENT_DATE AS CHAR(10)) ||
' ' || CAST(CURRENT_TIME AS CHAR(13))
```

Здесь для получения в символьном виде текущей даты и времени используется операция конкатенации строк. Функция CAST используется для преобразования одного типа данных к другому. Чтобы получить то же значение в типе данных TIMESTAMP, необходимо выполнить обратное преобразование типов данных:

```
CAST(CAST(CURRENT_DATE AS CHAR(10)) ||
' ' || CAST(CURRENT_TIME AS CHAR(13)) AS TIMESTAMP)
```

CURRENT_DATE типа DATE возвращает текущую дату сервера. Возвращаемое значение равно значению, полученному при помощи предварительно определенного литерала 'TODAY'.

CURRENT_TIME типа TIME возвращает текущее время сервера. Эта контекстная переменная возвращает не только время, но и тысячные доли секунды. В обращении к контекстной переменной CURRENT_TIME можно указывать и требуемые доли секунды:

```
CURRENT_TIME [(количество знаков в долях секунды>)]
```

Количество знаков может быть числом от 0 до 3. Если количество знаков не указано, предполагается 0 (в отличие от CURRENT_TIMESTAMP, где по умолчанию принимается значение 3).

Для получения текущего времени с точностью до тысячных долей секунды в символьном виде нужно выполнить преобразование следующего вида:

```
CAST(CURRENT_TIME(3) AS CHAR(20))
```

### CURRENT_CONNECTION

Контекстная переменная CURRENT_CONNECTION имеет тип данных INTEGER. Она возвращает число — системный идентификатор текущего соединения с базой данных. Такая переменная имеет смысл для программ, использующих при работе с базой данных программный интерфейс приложения API. Значение переменной хранится в странице заголовка базы и сбрасывается после restore. 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).

```
select CURRENT_CONNECTION from RDB$DATABASE;
```

### CURRENT_USER

CURRENT_USER типа VARCHAR(31) возвращает имя пользователя, который в настоящий момент соединен с базой данных.

```
SELECT post
FROM UserTable
WHERE UserName = CURRENT_USER;
```

### USER

USER — имя пользователя, связанного с текущим экземпляром клиентской библиотеки. Тип данных: VARCHAR(31). Это имя того же самого пользователя, которое может быть получено при обращении к контекстной переменной CURRENT_USER.

### CURRENT_ROLE

Контекстная переменная CURRENT_ROLE типа VARCHAR(31) возвращает имя роли, под которой с базой данных в настоящий момент соединился пользователь в операторе CONNECT при использовании предложения ROLE. Если при соединении с базой данных роль не была указана, то возвращается пустое значение NONE. Подробнее о назначении ролей для различных пользователей системы см. в документе «Руководство администратора».

```
select CURRENT_ROLE from RDB$DATABASE;
```

### CURRENT_TRANSACTION

Контекстная переменная CURRENT_TRANSACTION типа INTEGER возвращает число — системный идентификатор транзакции, под управлением которой выполняется текущий запрос. Значение CURRENT_TRANSACTION хранится в странице заголовка базы данных и сбрасывается в 0 после восстановления (или создания базы). Оно увеличивается при старте новой транзакции. Подробнее о транзакциях см. в главе 10 «Транзакции».

### ROW_COUNT

ROW_COUNT типа INTEGER указывает общее количество строк, которые были прочитаны, добавлены, изменены или удалены в процессе выполнения последнего оператора DML (INSERT, UPDATE, DELETE, SELECT или FETCH). Эта контекстная переменная может быть использована только в триггерах, хранимых процедурах или исполняемых блоках. Подробнее об использовании этой переменной см. в главе 11 «Хранимые процедуры, хранимые функции, триггеры и пакеты».

### SQLCODE, GDSCODE

Контекстные переменные SQLCODE и GDSCODE типа INTEGER позволяют получить значения соответствующих кодов ошибок базы данных. Эти контекстные переменные можно использовать в блоках WHEN GDSCODE, WHEN ANY, WHEN SQLCODE и WHEN EXCEPTION при условии, что код ошибки соответствует коду ошибки Ред Базы Данных. Вне обработчика ошибок GDSCODE и SQLCODE всегда равны 0. Вне PSQL не существуют вообще. Подробнее об использовании переменных SQLCODE и GDSCODE см. в главе 11 «Хранимые процедуры, хранимые функции, триггеры и пакеты».

### SQLSTATE

Контекстная переменная SQLSTATE типа CHAR(5) содержит 5 символов SQL-2003 — совместимого кода состояния, переданного оператором, вызвавшим ошибку. Вне обработчиков ошибок SQLSTATE всегда равен '00000', а вне PSQL не существует вообще. Эту контекстную переменную можно использовать в блоке WHEN ANY для проверки значения.

SQLSTATE предназначен для замены SQLCODE.

Любой код SQLSTATE состоит из двух символов класса 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.

### 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 (INSERT), изменения существующих данных (UPDATE) или удаления строк (DELETE). Эти переменные могут быть использоваться только в триггерах, обрабатывающих события базы данных. Подробнее об использовании этих переменных см. в главе 11 «Хранимые процедуры, хранимые функции, триггеры и пакеты».

### NEW и OLD

Контекстные переменные OLD.<columname> и NEW.<columname> доступны только в коде табличных триггеров. Более правильное название этих ключевых слов — префиксы имен столбцов. В триггерах можно обращаться к значению любого столбца таблицы (представления) до его изменения (операции обновления или вставки) 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 для всех видов триггеров является переменной только для чтения. Она недоступна в триггерах, вызываемых при добавлении данных, независимо от фазы события.

Контекстная переменная NEW в триггерах для фазы события после (AFTER) также является переменной только для чтения. Она недоступна в триггерах для события удаления данных.

Предварительно определенные литералы и контекстные переменные подробно описаны в приложении Ж «Контекстные переменные».

## 2.3 Преобразование типов данных. Функция CAST

Функция CAST позволяет преобразовывать данные из одного типа данных в другой, допустимый для исходного значения. Синтаксис функции показан в листинге 2.1.

**Листинг 2.1. 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> }
```

Преобразование константы NULL в любой тип данных всегда дает NULL.

Хотя в некоторых случаях сервер Ред База 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.

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

Тип данных `BLOB` подтипа `TEXT` также допускает преобразования (но с максимальным размером 32765 байт).

## Преобразование в целочисленные типы данных

В целочисленные типы данных (`SMALLINT, INTEGER, BIGINT`) можно выполнять преобразование целочисленных данных, числовых данных и констант с фиксированной точкой (`DECIMAL, NUMERIC`), чисел и констант с плавающей точкой (`FLOAT, DOUBLE PRECISION`), данных текстового `BLOB` и строковых данных (`CHAR, VARCHAR, NCHAR, NCHAR VARYING`), содержащих только цифры, десятичную точку, знак числа (+ или −) и признак показателя степени (букву $E$ или $e$). Если исходное число в строке или в дробном числе содержит дробные знаки, то они просто отбрасываются, округление не выполняется. Преобразуемое число не должно выходить за пределы диапазона допустимых чисел для типа, в который происходит преобразование. Иначе будет выдано исключение (арифметическое переполнение).

Например, следующее преобразование дробного числа с фиксированной точкой даст результат 12300:

```
CAST(12300.45 AS SMALLINT)
```

Преобразование числа с плавающей точкой также происходит правильно (результатом будет 1230):

```
CAST(12300.45E-1 AS SMALLINT)
```

Допустимо и такое преобразование строк в целочисленный тип данных:

```
CAST('12300.45E-1' AS SMALLINT)
```

Здесь строковый литерал имеет вид правильного числа с плавающей точкой. Преобразование дает также число 1230.

Однако такое преобразование как

```
CAST(12300000.45 AS SMALLINT)
```

вызывает исключение — арифметическое переполнение, поскольку в типе данных `SMALLINT` не может храниться положительное число, превышающее 32767.

Преобразование даты и/или времени в целочисленный тип данных невозможно. Попытка выполнить такую операцию приводит к исключению «ошибка преобразования».

## Преобразование в дробные числа с фиксированной точкой

В дробные числа с фиксированной точкой (`DECIMAL, NUMERIC`) можно преобразовывать все перечисленные целочисленные данные и данные с фиксированной или плавающей точкой, данные типа `BLOB` подтипа `TEXT`, а также и «правильные» строки, содержащие данные, по форме соответствующие числам (целым, с фиксированной или с плавающей точкой). Преобразование даты и времени, как и в случае целочисленных данных, невозможно.

## Преобразование в строковые типы данных

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. Иначе будет получено исключение — усечение строки. Например, следующее преобразование пройдет правильно:

```
CAST(CURRENT_TIME AS CHAR(13))
```

А такое даст исключение, сообщающее об усечении строки, поскольку для представления времени требуется не менее 13 символов, включая четыре знака после десятичной точки в указании секунд:

```
CAST(CURRENT_TIME AS CHAR(10))
```

## Преобразование в типы данных даты и времени

В тип данных `DATE` можно преобразовать любую «правильную» строку, содержащую дату в одном из допустимых форматов (см. далее в этой главе). Например, допустимы такие преобразования:

```
CAST('14.07.2007' AS DATE)
CAST('07-14-2007' AS DATE)
CAST('07/14/2007' AS DATE)
CAST('2007-07-14' AS DATE)
CAST('2007/07/14' AS DATE)
CAST('2007.07.14' AS DATE)
CAST('14-JUL-2007' AS DATE)
```

Все они дадут одно и то же значение даты: 14 июля 2007 г.

Строки можно преобразовывать во время. Часы, минуты и секунды отделяются друг от друга двоеточием. Например, допустимо такое преобразование:

```
CAST('23:18:14' AS TIME)
```

Допустимо также использование в качестве разделителей минут, часов и секунд символов точки вместо двоеточия. Следующий вариант преобразования аналогичен предыдущему:

```
CAST('23.18.14' AS TIME)
```

Не допускается преобразование времени в дату, равно как и даты во время. Здесь выдается исключение «ошибка преобразования».

Тип данных `TIMESTAMP` (хранит дату и время) можно преобразовать как в `DATE`, так и в `TIME`. В первом случае в результате преобразования отбрасывается время, во втором — дата.

Тип данных `DATE` при преобразовании в `TIMESTAMP` будет содержать правильную дату и нулевое значение времени — `'00:00:00.0000'`. Например, можно выполнить такое преобразование:

```
CAST('14-JUL-2007' AS TIMESTAMP)
```

Результатом будет дата 14 июля 2007 г. и время 00:00:00.0000.

Преобразование `TIME` в `TIMESTAMP` невозможно. Такая попытка вызывает исключение «ошибка преобразования».

## Преобразование в тип данных BLOB

В текстовый `BLOB` можно преобразовывать данные любого типа.

### Преобразование в логический тип данных

В логический тип данных `BOOLEAN` можно преобразовать строковые данные и данные типа `BLOB`.

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

## 2.4 Числовые типы данных

В SQL существует достаточно большое количество числовых типов данных. В эту категорию типов данных входят числа с фиксированной точкой (целочисленные — `SMALLINT, INTEGER` и `BIGINT`, дробные — `DECIMAL, NUMERIC`) и числа с плавающей точкой (`FLOAT, DOUBLE PRECISION`). Числа с фиксированной точкой (целые и дробные) иногда называют также точными числами (`exact numeric`).

Для числовых типов данных определены четыре арифметические операции — сложение, вычитание, умножение и деление.

Следует заметить, что в диалекте 3 базы данных типы данных `DECIMAL` и `NUMERIC` хранятся и используются совершенно одинаковым образом. Способ хранения таких данных зависит от разрядности — общего количества знаков, отводимых под число типа `DECIMAL` или `NUMERIC`. При разрядности до 4 знаков число хранится как `SMALLINT`. От 5 до 9 — как `INTEGER`. От 10 до 18 — как `BIGINT`.

Операции сложения и вычитания для всех числовых типов данных выполняются обычным образом.

Следует только быть особенно внимательными при выполнении операций умножения и деления чисел с фиксированной точкой (`SMALLINT, INTEGER, BIGINT, DECIMAL` и `NUMERIC`). В этих операциях результат будет иметь количество дробных знаков, равное сумме дробных знаков обоих операндов.

Классический пример — деление двух целых чисел. Результатом операции

```
1 / 3
```

будет 0, потому что сумма дробных знаков операндов равна нулю, а целая часть в результате выполнения операции деления будет равна нулю. Результат является целочисленным. Чтобы получить значение с заданной точностью, необходимо у одного или у обоих операндов явно указать нужное количество нулевых дробных знаков. Например, операция

```
1.00 / 3
```

вернет уже более верное число — 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 чисел с плавающей точкой все операции выполняются таким же образом, как принято во всех языках программирования. Следует помнить, что эти типы данных имеют конкретное количество значащих цифр (15 для `DOUBLE PRECISION` и 7 для `FLOAT`). Если, например, к очень большому числу с плавающей точкой прибавить очень маленькое число (или вычесть из него такое число), то результат не будет отличаться от первого, большего, числа.

В следующем выражении

```
1.23E+20 - 1.23E-24
```

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 следующие полезные агрегатные функции, определенные в SQL — `MIN, MAX, SUM, AVG и др.`. Функции возвращают единственное значение, используя множество входных данных, получаемых, как правило, из оператора `SELECT`. Есть также агрегатная функция подсчета количества значений `COUNT`, возвращающая целое число, но не связанная в качестве входных параметров с числовым типом данных. Подробнее об этих и некоторых других функциях см. далее в этой главе.

К числовым типам данных применимо множество встроенных в SQL функций. Описание встроенных функций см. в приложении Е «Функции».

## 2.5 Строковые типы данных

К строковым (символьным) типам данных относятся следующие типы данных, определенные в SQL Ред База Данных: `CHAR (CHARACTER), VARCHAR (VARYING CHARACTER), NCHAR (NATIONAL CHARACTER)` и `NCHAR VARYING (NATIONAL CHARACTER VARYING)`.

Строковые константы заключаются в апострофы (в ранних версиях InterBase, до версии 6.0, допускалось использование и кавычек). Если в строковой константе присутствует символ апостроф, то он должен быть представлен двумя подряд идущими апострофами.

Для строковых типов данных определена только одна 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 и ряда других типов данных применимо множество встроенных функций SQL. В этой главе будут рассмотрены функции `TRIM, SUBSTRING, LEFT, RIGHT, UPPER, LOWER, CHARACTER_LENGTH, OCTET_LENGTH, BIT_LENGTH`. Описание других функций см. в приложении Е «Функции».

### Функции для строковых типов данных

#### Функция 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 представлен в листинге 2.2.

**Листинг 2.2. 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.

### Функция SUBSTRING

Встроенная функция `SUBSTRING` возвращает подстроку исходной строки. Синтаксис функции представлен в листинге 2.3.

**Листинг 2.3. 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 строка имеет тип BLOB, то и результат будет иметь тип BLOB. Для любых других типов результатом будет типVARCHAR(n). Для строки, не являющейся BLOB, длина результата функции всегда будет равна длине исходной строки,независимо от значений других параметров.

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.

**Примеры.** Выполнение функции

```
SUBSTRING ('Руководство ' FROM 5 FOR 3)
```

даст строку «вод». Следующая функция

```
SUBSTRING ('Руководство ' FROM 5 FOR 100)
```

вернет строку «водство». Здесь происходит усечение результата без выдачи диагностических сообщений.

Выполнение функции

```
SUBSTRING ('123456' FROM 8 FOR 10)
```

вернет пустую строку, не содержащую никаких символов.

Выполнение функции

```
SUBSTRING('abcdefg' SIMILAR '_#"%#"_' ESCAPE '#' )
```

вернет строку «bcdef».

### Функции LEFT и RIGHT

Упрощенным вариантом этой функции являются функции LEFT, которая возвращает указанные первые символы строки, и RIGHT, возвращающая последние символы строки.

**Листинг 2.4. Синтаксис функций LEFT и RIGHT**

```
LEFT (<строка>, <длина подстроки>)
RIGHT (<строка>, <длина подстроки>)
```

Функции поддерживают текстовые BLOB любой длины и с любыми наборами символов. Если строка имеет тип BLOB, то и результат будет иметь тип BLOB. Для любых других типов результатом будет типVARCHAR(n), где n является длиной исходной строки.

Если числовой параметр превысит длину текста, результатом будет исходный текст.

### Функции UPPER и LOWER

Функция UPPER переводит все буквы исходной строки в верхний регистр. Функция правильно работает не только с латинскими буквами, но и с буквами кириллицы. Синтаксис функции см. в листинге 2.5

**Листинг 2.5. Синтаксис функции перевода букв строки в прописные UPPER**

```
UPPER (<строка>)
```

Выполнение функции

```
UPPER ('россия')
```

вернет строку «РОССИЯ».

Функция LOWER переводит все буквы исходной строки в нижний регистр. Функция правильно работает с латинскими буквами и с буквами кириллицы. Синтаксис функции см. в листинге 2.6.

**Листинг 2.6. Синтаксис функции перевода букв строки в строчные LOWER**

```
LOWER (<строка>)
```

Выполнение функции

```
LOWER ('РОССИЯ')
```

вернет строку «россия».

Точный результат зависит от набора символов входной строки. Например, для наборов символов NONE и ASCII только ASCII символы переводятся в верхний(нижний) регистр; для OCTETS — вся входная строка возвращается без изменений.

Функции поддерживают тип данных BLOB.

### Функции CHARACTER_LENGTH, OCTET_LENGTH и BIT_LENGTH

Три функции позволяют определить размер строки — CHARACTER_LENGTH, OCTET_LENGTH и BIT_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 — см. далее) будет больше, чем количество символов.

Синтаксис функции CHARACTER_LENGTH представлен в листинге 2.7.

**Листинг 2.7. Синтаксис функции подсчета количества символов во входном параметре CHARACTER_LENGTH**

```
{CHARACTER_LENGTH | CHAR_LENGTH} (<строка>)
```

Функция OCTET_LENGTH возвращает количество байтов, занимаемых входным параметром функции. Не для всех наборов символов возвращаемое значение равняется значению CHARACTER_LENGTH. Синтаксис функции представлен в листинге 2.8.

**Листинг 2.8. Синтаксис функции подсчета количества байтов во входном параметре OCTET_LENGTH**

```
OCTET_LENGTH (<строка>)
```

Функция BIT_LENGTH возвращает количество битов во входном параметре. Возвращаемое значение будет в точности равно OCTET_LENGTH $*$ 8. Синтаксис функции показан в листинге 2.9.

**Листинг 2.9. Синтаксис функции подсчета количества битов во входном параметре BIT_LENGTH**

```
BIT_LENGTH (<строка>)
```

Поскольку для строк определена операция сравнения, к строкам могут также применяться и агрегатные функции MIN и MAX. Эти функции будут отыскивать, соответственно, минимальное и максимальное значение указанных столбцов выбранных строк таблицы, представления или хранимой процедуры выбора. Сравнение строк осуществляется в соответствии с используемым для столбца набором символов (CHARACTER SET) и порядком сортировки (COLLATION ORDER). Для правильного сравнения строк, которые содержат буквы кириллицы, следует использовать набор символов WIN1251 и порядок сортировки PXW_CYRL.

Существуют другие строковые функции.

- POSITION отыскивает позицию подстроки в строке.
- REVERSE переписывает символы строки в обратном порядке.
- REPLACE отыскивает в строке заданную подстроку и заменяет ее на другую.
- LPAD добавляет к строке слева указанную подстроку.
- RPAD добавляет к строке справа указанную подстроку.
- OVERLAY заменяет указанное количество символов на заданное значение.
- HASH возвращает хэш-значение входной строки.
- ASCII_CHAR возвращает ASCII символ соответствующий номеру, переданному в качестве аргумента.
- ASCII_VAL возвращает ASCII код символа, переданного в качестве аргумента.

## 2.6 Логический тип данных

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.

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

**Пример.** Приведем 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>
```

# 2.7 Типы данных даты и времени

Существует три типа данных для представления даты и времени — DATE, TIME и TIMESTAMP, позволяющие хранить, соответственно, дату, время и объединение даты и времени.

Подробное описание функций, применяемых к типам данных даты и времени, см. в приложении E «Функции».

## Тип данных DATE

Этот тип данных позволяет хранить только дату в диапазоне от 1 января 1 года до 31 декабря 9999 года.

Для литералов, представляющих дату, в SQL существует много форматов. При описании синтаксиса для формата типа DATE 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). Для номера дня и номера месяца ведущий ноль можно не указывать. Год может быть

задан и числом с меньшей, чем четыре, значимостью. Вот основные форматы даты, используемые в SQL Ред База Данных:

**Листинг 2.10. Основные форматы даты**

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

`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.

Например, ту же дату 14 июля 2007 года можно записать в следующем виде (что принято в нашей стране):

```
'14.07.2007'
```

или же

```
'14-JUL-2007', '07-12-2007', '07/12/2007' и т.д.
```

На сегодняшний день все числа года от 0 до 50, указывают на годы, которые начинаются с 2000. Большие: 51 и выше — годы с 1900. Для устранения неопределенности и возможных изменений в будущих версиях системы можно рекомендовать задание полного номера года: например, 14 июля 2007 года следует записать в виде '14.07.2007', а 14 июля 1907 года — в виде '14.07.1907'.

Чтобы задать более ранние даты, следует всегда в номере года указывать все ведущие нули. Например, чтобы указать 1 января 1 года, нужно записать:

```
'1.1.0001' или '01.01.0001'
```

## Тип данных TIME

Тип данных время (`TIME`) позволяет хранить только время с точностью до десятитысячной доли секунды (до 100 микросекунд). Он 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 варианта, где вместо двоеточия в качестве разделителя используется точка:

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

Например, время 23 часа, 16 минут и 32 секунды с 98 десятитысячными долями секунды нужно записать в виде:

```
'23:16:32.0098'
```

или

```
'23.16.32.0098'
```

В одном литерале можно использовать в качестве разделителя и двоеточие, и точку.

Чтобы представить переменную VARIABLE_T типа данных TIME в виде, удобном для восприятия, следует выполнить преобразование:

```
CAST(VARIABLE_T AS CHAR(13))
```

Для строкового литерала следует выполнить такое преобразование:

```
CAST(CAST('23.16.32.0098' AS TIME) AS CHAR(13))
```

Во втором примере выполняется преобразование строки к типу TIME, а полученный результат преобразуется обратно к строковому виду, понятному для человека.

## Тип данных TIMESTAMP

Тип данных дата и время (TIMESTAMP) представляет собой соединение даты и времени, которые в литералах просто разделяются любым количеством пробелов.

Например, для задания 12 часов 30 минут 14 июня 2007 года можно записать:

```
CAST(CAST('14.07.2007 12:30' AS TIMESTAMP) AS CHAR(24))
```

## Арифметические операции для типов данных даты и времени

Для типов данных даты (DATE) и времени (TIME) определены операции сложения и вычитания.

Операция сложения для типа DATE и целого числа дает дату, увеличенную на заданное количество дней. Вычитание из типа данных DATE целого числа возвращает дату, уменьшенную на указанное количество дней. В операции можно указать и дробное число. К ошибке это не приводит, происходит правильное округление числа до ближайшего целого.

Например, следующая операция дает завтрашнюю дату:

```
CURRENT_DATE + 1
```

Чтобы получить вчерашнюю дату, нужно записать:

```
CURRENT_DATE - 1
```

Вчерашнюю дату можно получить, также записав:

```
CURRENT_DATE - 1.003
```

Вычитание двух дат дает количество дней в интервале. Например, чтобы узнать, сколько дней осталось до Нового 2009 года, нужно записать:

```
CAST('31.12.2008' AS DATE) - CURRENT_DATE
```

Сложение «TIME + число» дает указанное время, увеличенное на заданное число секунд, включая десятитысячные доли секунды. Здесь в операции можно использовать дробное число.

Соответственно, вычитание «TIME – число» дает время, уменьшенное на заданное число секунд, включая десятитысячные доли секунды.

Вычитание двух переменных типа TIME дает интервал времени в секундах (включая десятитысячные доли секунды).

Для типа данных TIMESTAMP ни одна из перечисленных операций недопустима.

## Функции для типов данных даты и времени

### Функция EXTRACT

Для типов данных даты (DATE), времени (TIME) и даты/времени (TIMESTAMP) может использоваться встроенная функция EXTRACT, позволяющая выделять различные элементы даты и времени. Ее синтаксис представлен в листинге 2.11.

**Листинг 2.11. Синтаксис функции выделения элементов даты и времени EXTRACT**

```
EXTRACT (<выделяемый элемент> FROM <исходное данное>)
```

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.

Выделять часы, минуты и секунды можно лишь в типах данных, содержащих время: TIME и TIMESTAMP. Аналогично, 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)
```

Для типов данных DATE и TIME допустимы также и функции MIN и MAX, возвращающие, соответственно, минимальное или максимальное значение в группе столбцов из выбранных оператором SELECT строк.

### Функция DATEADD

Функция DATEADD позволяет изменить значение заданной даты и/или времени. 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 (см. листинг 2.12).

**Листинг 2.12. 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 требует явного преобразования литералов к соответствующему типу данных.

**Пример.** Чтобы получить завтрашнюю дату, нужно вызвать следующую функцию:

```
DATEADD(DAY, 1, CURRENT_DATE)
```

### Функция DATEDIFF

Функция DATEDIFF возвращает целое число, задающее интервал в соответствии с указанным элементом между двумя значениями типа данных DATE, TIME или TIMESTAMP. У функции есть два формата — см. листинг 2.13.

**Листинг 2.13. 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))
```

## 2.8 Тип данных 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 2.14. Синтаксис объявления типа BLOB**

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

Также можно использовать сокращенный синтаксис:

**Листинг 2.15. 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 База Данных существуют семь заранее предопределенных подтипов. Они представлены в таблице 2.2. Их не следует использовать для каких-то своих внутренних целей.

Таблица 2.2 — Предопределенные подтипы типа BLOB

| Подтип BLOB | Имя подтипа | Назначение |
|---|---|---|
| 0 | BINARY | Неструктурированные данные или данные неопределенного типа |
| 1 | TEXT | Текстовые данные |
| 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 расшире-

ниях. На клиентских программах лежит ответственность за то, что в поля BLOB заданного подтипа записываются данные соответствующего вида.

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.

## 2.9 Тип данных 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.

Глава 3

# Работа с базой данных

Прежде, чем начать создавать объекты базы данных и заполнять базу данными конкретной предметной области, необходимо создать базу данных с необходимыми характеристиками.

## 3.1 Создание базы данных

Для создания базы данных используется оператор SQL `CREATE DATABASE`. Его синтаксис в нотациях Бэкуса-Наура представлен в листинге 3.1.

**Листинг 3.1. Синтаксис оператора создания базы данных `CREATE DATABASE`**

```
CREATE {DATABASE | SCHEMA} '<спецификация файла>'
  [USER '<имя пользователя>' [PASSWORD '<пароль>']]
  [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>...]]
  [[SET] MAC PLUGIN <модуль мандатного доступа>];

<спецификация файла> ::=
  [{ <имя сервера>: | \\<имя сервера>\ }] { <путь к файлу БД> | <алиас БД> }

<вторичный файл> ::=
  FILE '<спецификация файла>'
  [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`.

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. Файл может иметь любое расширение или не иметь вообще никакого расширения. Для файлов СУБД Ред База Данных принято использовать расширение `fdb`.

Например, в операционной системе Windows можно указать спецификацию файла в следующем виде:

```
'D:\RedSoftDatabase\work.fdb'
```

Если файл создается на сервере в локальной сети, то в любом варианте конфигурации Windows путь к базе данных всегда можно задать для протокола TCP/IP. 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 операционная система поддерживает также и протокол под названием именованные каналы (Named Pipes), то аналогичный путь к файлу базы данных можно задать следующим образом:

```
'\\Server\D:\RedSoftDatabase\work.fdb'
```

В операционной системе Linux путь к файлу задается несколько иначе, например,

```
'/home/share/RedSoftDatabase/work.fdb'
```

Для создания базы данных на другом компьютере в локальной сети при использовании операционной системы Linux нужно в этом случае указать и имя сервера:

```
'ServerL:/home/share/RedSoftDatabase/work.fdb'
```

Вместо полного пути к файлу базы можно использовать псевдонимы (aliases). Псевдонимы описываются в файле `databases.conf`.

Необязательные предложения в операторе создания базы данных `USER` и `PASSWORD` задают, соответственно, имя и пароль пользователя, присутствующего в базе данных безопасности `security3.fdb`. Файл базы данных безопасности находится в корневом каталоге инсталляции Ред База Данных. Заданный в предложении `USER` пользователь становится владельцем созданной базы данных и имеет к ней неограниченные полномочия.

Имя пользователя может занимать до 31 байта. Оно нечувствительно к регистру. Пароль пользователя чувствителен к регистру. Он может содержать до 32 символов, однако только первые восемь имеют значение.

После инсталляции Ред База 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.

Если у вас установлены соответствующие значения переменных окружения `ISC_USER` и `ISC_PASSWORD`, то в операторе `CREATE DATABASE/SCHEMA` вы можете не указывать предложений `USER` и/или `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 `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 (значение по умолчанию), 8192 и 16384. Если вы зададите неправильное значение размера страницы, то система не выдаст сообщения об ошибке, а установит размер до ближайшего меньшего числа. Если указать значение меньше чем 4096, то будет выбрано значение по умолчанию — 4096.

> В более ранних версиях системы управления базами данных могли применяться и меньшие размеры страниц — 1024 и 2048, однако их использование на практике оказалось весьма неэффективным. Часто увеличение размера страницы базы данных повышает производительность системы, скорость выполнения запросов, особенно если используются поля `BLOB` больших размеров, множество сложных индексов, база данных содержит много данных, строки таблиц содержат большое количество столбцов больших размеров.

Предложение `LENGTH` задает максимальный размер первичного или вторичного файла базы данных в страницах. При создании базы данных любой файл (первичный или вторичный) независимо от зна-

чения 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.

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 приложении В «Наборы символов и порядки сортировки».

Необязательный параметр COLLATION, связанный с набором символов, позволяет создавать все текстовые столбцы, домены и переменные с указанной последовательностью сортировки, если не указан другой COLLATE.

> COLLATION, используемый по умолчанию для набора символов в базе данных, может быть изменен с помощью ALTER CHARACTER SET.

Ключевое слово DIFFERENCE FILE служит параметром для задания файла. Данный параметр позволяет задать имя дельта-файла, который будет создаваться при выполнении команды ALTER DATABASE BEGIN BACKUPили запуске утилиты nBackup из командной строки.

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. далее 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.

Для активации мандатного доступа при создании базы указывается предложение [SET] MAC PLUGIN, которое задает необходимый модуль мандатного доступа. При подключении к базе указанный модуль загружается сервером и используется для контроля доступа. Более подробно о мандатном принципе контроля доступа см. в Руководстве администратора.

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 имеет смысл использовать диалект 3, который позволяет более эффективно использовать внешнюю память и средства системы управления базами данных Ред База Данных.

Средствами операторов SQL можно лишь добавить новые вторичные файлы к существующей базе данных — см. оператор `ALTER DATABASE`. Другие характеристики изменить таким образом невозможно. Размер страницы, количество и размеры вторичных файлов можно изменять, выполняя резервное копирование и восстановление базы данных. Подробности см. в документе «Руководство администратора».

## 3.2 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 = 16384
  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 — 16384.

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 5 «Работа с таблицами».

В `isql` можно отобразить состояние созданной базы данных, соединившись с базой данных (для этого используется оператор `CONNECT` — см. далее в этой главе в разделе 3.3 «Соединение с существующей базой данных») и выполнив оператор `isql SHOW DATABASE`:

```
SQL> SHOW DATABASE;
-------------------------------------------
Database: D:\RedSoftDatabase\work.fdb
Owner: WIZARD
PAGE_SIZE 16384
Number of DB pages allocated = 140
Sweep interval = 20000
Forced Writes are ON
Transaction - oldest = 6
Transaction - oldest active = 7
Transaction - oldest snapshot = 7
Transaction - Next = 10
ODS = 11.2
Default Character set: WIN1251;
```

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 '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 $4096 \times 10000$ 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:

```
SQL> SHOW DATABASE;
-----------------------------------------------
Database: D:\RedSoftDatabase\work.fdb
Owner: WIZARD
File 1: 'D:\RedSoftDatabase\work.fd2', length 10000, start 10001
File 2: 'D:\RedSoftDatabase\work.fd3', length 0, start 20001
PAGE_SIZE 4096
Number of DB pages allocated = 168
Sweep interval = 20000
Forced Writes are ON
Transaction - oldest = 1
Transaction - oldest active = 2
Transaction - oldest snapshot = 2
Transaction - Next = 6
ODS = 11.2
Default Character set: WIN1251;
```

**Пример 3.** Точно такую же базу данных мы получим, если зададим оператор следующим образом:

```
SET SQL DIALECT 3;
CREATE DATABASE '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 '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 '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;
```

## 3.3 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`. Его синтаксис представлен в листинге 3.2.

**Листинг 3.2. 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 31 символа. 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 приложении В.

Синтаксис оператора `SET SQL DIALECT` (листинг 3.3):

**Листинг 3.3. Синтаксис оператора задания диалекта клиента `SET SQL DIALECT`**

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

Синтаксис оператора `SET NAMES` см. в листинге 3.4:

**Листинг 3.4. 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 'D:\RedSoftDatabase\work.fdb'
  USER 'wizard' PASSWORD 'master';
```

При работе из командной строки DOS оператор `SET NAMES` дложен быть записан в следующем виде:

```
SET NAMES DOS866;
```

## 3.4 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 3.5:

**Листинг 3.5. Синтаксис оператора изменения базы данных ALTER DATABASE**

```
ALTER {DATABASE | SCHEMA}
[ADD <вторичный файл> [ADD <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]
[[SET] MAC PLUGIN <модуль мандатного доступа> | DROP MAC PLUGIN];
```

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 3.1 «Создание базы данных» этой главы).

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`).

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.

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.

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.

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`.

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.

Для активации или изменения модуля мандатного доступа у существующей базы указывается предложение `[SET] MAC PLUGIN`, которое задает необходимый модуль мандатного доступа. После этого следует перезапустить все подключения. Для отключения контроля доступа в предложении `DROP MAC PLUGIN` не указывается никакой модуль. Более подробно о мандатном принципе контроля доступа см. в Руководстве администратора.

Изменять базу данных может ее владелец, пользователь с административными привилегиями и пользователь с привилегией `ALTER DATABASE`.

> Оператор `ALTER DATABASE` позволяет лишь добавлять к существующей базе данных дополнительные вторичные файлы. Изменить размер страницы, набор символов по умолчанию для строковых данных 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».

**Пример.** Для добавления еще одного вторичного файла к существующей базе данных, созданной в одном из примеров раздела 3.1, нужно выполнить следующие операторы:

```
SET SQL DIALECT 3;
SET NAMES WIN1251;
CONNECT 'D:\RedSoftDatabase\work.fdb'
  USER 'wizard' PASSWORD 'master';
ALTER DATABASE
```

```
ADD FILE 'D:\RedSoftDatabase\work.fd4'
  STARTING AT PAGE 30001;
```

## 3.5 Удаление базы данных

Для удаления существующей базы данных, с которой выполнено в настоящий момент соединение, используется оператор SQL DROP DATABASE (листинг 3.6):

**Листинг 3.6. 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. далее), 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 (см. раздел 3.7 «Использование оперативных копий» далее в этой главе).

**Пример.** Следующие операторы позволяют удалить существующую базу данных. Вначале выполняется соединение с базой данных, затем эта база данных удаляется:

```
CONNECT 'D:\RedSoftDatabase\work.fdb'
  USER 'wizard' PASSWORD 'master';
DROP DATABASE;
```

Если другой пользователь, отличный от владельца базы данных или пользователя SYSDBA, соединившись с базой данных, попытается удалить эту базу, то будет выдано сообщение об ошибке:

```
This user does not have privilege to perform this operation on this object.
no permission for drop access to database D:\RedSoftDatabase\work.fdb
(Этот пользователь не имеет привилегии для выполнения этой операции с данным
объектом. Нет полномочий для удаления базы данных...).
```

## 3.6 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 3.7.

**Листинг 3.7. 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 '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 'D:\RedSoftDatabase\work.fdb'
  USER 'wizard' PASSWORD 'master';
COMMENT ON DATABASE IS '<Новый текст примечания>';
```

## 3.7 Использование оперативных копий

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 дан-

ных 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.

## 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 3.8:

**Листинг 3.8. Синтаксис оператора создания оперативной копии `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`. Подробнее см. в документе «Руководство администратора».

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.

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.

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.

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.

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.

Размер страницы оперативной копии автоматически устанавливается таким же, что и в исходной базе данных, и не может быть изменен.

**Пример 1.** Чтобы создать для базы данных однофайловую оперативную копию, нужно выполнить следующий оператор:

```
connect '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 опе-

раторы:

```
connect '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.

## Удаление оперативной копии

Для удаления существующей оперативной копии используется оператор `SQL DROP SHADOW`. Его синтаксис представлен в листинге 3.9:

**Листинг 3.9. 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 пользователем с привилегией `ALTER DATABASE`. Подробнее см. в документе «Руководство администратора».

Глава 4

# Работа с доменами

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. Основной характеристикой домена является тип данных.

## 4.1 Создание домена

Для создания нового домена в базе данных используется оператор `CREATE DOMAIN`. Синтаксис оператора представлен в листинге 4.1:

**Листинг 4.1. 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>] ];
```

Имя домена должно быть уникальным среди имен доменов базы данных. Имя не может превышать 31 символа.

Создавать домен может администратор и пользователь с привилегией `CREATE DOMAIN`.

### Задание типа данных

Тип данных задается следующей синтаксической конструкцией (листинге 4.2):

**Листинг 4.2. Синтаксис задания типа данных**

```
<тип данных> ::= {
    {SMALLINT | INTEGER | BIGINT} [<размерность массива>]
  | BOOLEAN [<размерность массива>]
  | {FLOAT | DOUBLE PRECISION} [<размерность массива>]
  | {DATE | TIME | TIMESTAMP} [<размерность массива>]
  | {DECIMAL | NUMERIC} [(<точность> [ ,<масштаб>])] [<размерность массива>]
  | {CHAR | CHARACTER | CHARACTER VARYING | VARCHAR} [(<размер>)]
      [CHARACTER SET <набор символов>] [<размерность массива>]
  | {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 любого типа данных, кроме BLOB, можно указать и размерность массива, если этот элемент является массивом. Для массива задается начальный номер элемента в массиве (неотрицательное число «целое 1») и через двоеточие последний номер («целое 2»). Если «целое 1» в этой синтаксической конструкции не указано, то предполагается значение единица. Если массив многомерный, то через запятую указываются и другие пары элементов. Размерность задается в квадратных скобках.

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.

Тип данных в синтаксисе оператора создания домена — один из допустимых, ранее описанных типов данных. Это единственный обязательный параметр в операторе создания домена.

При описании символьного домена и домена типа BLOB можно в предложении CHARACTER SET указать набор символов, если требуется набор, отличный от набора символов по умолчанию, установленный в операторе CREATE DATABASE для всей базы данных. Если для домена с типом данных BLOB указан подтип, то для такого домена нельзя задавать набор символов. Считается, что он предопределен подтипом. Кроме того, можно в предложении COLLATE задать и порядок сортировки (для типа данных BLOB использование COLLATE недопустимо).

## Значение по умолчанию

Предложение DEFAULT задает значение по умолчанию — что должно быть помещено в столбец таблицы, основанный на этом домене, если пользователь в операторе INSERT, добавляющем данные в таблицу, не укажет значение для этого столбца. Значение по умолчанию не используется при выполнении оператора изменения данных в таблице UPDATE. Если в этом операторе пользователь не укажет значение для конкретного столбца, то это значение просто не изменяется.

Значением по умолчанию может быть литерал, пустое значение NULL. Литералом может быть любая самоопределенная константа соответствующего типа, предварительно определенный литерал или контекстная переменная. Если значение по умолчанию не устанавливается, то подразумевается пустое значение NULL. В значении по умолчанию нельзя задавать выражения.

## 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.

## Синтаксис ограничения домена

Предложение CHECK, являясь ограничением домена, задает некоторое условие (заключаемое в круглые скобки), которому должно удовлетворять значение, помещаемое оператором INSERT в столбец, основанный на этом домене, или изменяемое оператором UPDATE для некоторой существующей строки столбца таблицы, основанного на этом домене. Предложение неприменимо к доменам типа BLOB.

Условие в предложении CHECK также иногда называется предикатом. Это логическое выражение, которое может возвращать значения TRUE (истина), FALSE (ложь) и UNKNOWN (неопределенное, неизвестное значение), если в операции принимает участие столбец, имеющий пустое значение NULL.

Условие считается выполненным, если этот предикат возвращает значение TRUE.

Условие домена может быть достаточно сложным. Его синтаксис показан в листинге 4.3.

**Листинг 4.3. 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>)
  | EXISTS (<поиск многих>)
  | 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>
}
```

Для того чтобы условие выполнялось, выражение должно возвращать значение только TRUE (истина). Во многих случаях, когда некоторые значения в условии содержат пустое значение NULL, выражение может возвращать и значение UNKNOWN (неопределенное), что не позволит поместить новое значение в соответствующий столбец (или изменить существующее значение столбца).

## Оператор сравнения

Синтаксис оператора сравнения представлен в листинге 4.4.

**Листинг 4.4. Синтаксис оператора сравнения для домена**

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

Оператор сравнения задает проверку равенства, неравенства и т.д. различных значений. В операторе сравнения символы «!» и «^» означают отрицание. Оператор может применяться к любому типу данных, кроме BLOB. При необходимости можно выполнить явное преобразование типа у операндов сравнения, используя функцию CAST.

| **Операторы** | = | <> , != , ^= | > | < | >= , !< , ^< | <= , !> , ^> |
|---|---|---|---|---|---|---|
| **Значение** | Равно | Не равно | Больше | Меньше | Больше или равно, не меньше | Меньше или равно, не больше |

Результатом сравнения, когда один из операндов или оба имеют значение NULL, будет UNKNOWN, то есть условие не выполняется, так же как и в случае, когда предикат возвращает значение FALSE.

Символьный тип данных можно сравнивать с любым типом данных, кроме BLOB. В таких операциях сравнения обычно осуществляется неявное преобразование других типов данных к символьному типу. Лучшим же вариантом в операции сравнения операндов с различными типами данных

является явное преобразование их типов данных к символьному типу с использованием функции CAST. Во многих случаях это позволит избежать серьезных ошибок.

Сравнение числовых данных между собой никогда не вызывает исключений. Например, можно сравнивать целочисленный тип данных с дробным 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. Дату или время можно сравнивать с символьным типом данных, если строка содержит дату или время в «правильном» виде; при этом рекомендуется все же выполнить явное преобразование сравниваемой строки к соответствующему типу, используя функцию CAST.

Нельзя дату сравнивать со временем.

## Значение в условии домена

Значением в условии домена может быть литерал, предварительно определенный литерал, контекстная переменная, а также любое правильное выражение с использованием допустимых операндов. В качестве любого операнда может быть использован оператор SELECT, заключенный в круглые скобки и возвращающий одно значение или NULL. Для построения выражений можно использовать как встроенные функции SQL, так и функции, определенные пользователем (UDF). Подробнее об использовании UDF см. в приложении Г «Функции, определенные пользователем (UDF)». Выражения (значения) могут присутствовать как в левой, так и в правой части языковых конструкций условий домена.

Формально синтаксис значения в условии домена описывается следующим образом (см. листинг 4.5):

**Листинг 4.5. Синтаксис значения в условии домена**

```
<значение> ::= {
    VALUE [[<элемент массива>]]
  | <литерал>
  | <выражение>
  | 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> ...])]
  | NULL
}
```

### 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>))
```

**Литералы и выражения**

Литерал — это числовая константа, строковая константа, заключенная в апострофы, литерал даты или времени, предварительно определенный литерал, контекстная переменная.

Выражение — любое правильное выражение SQL. Это может быть арифметическое, строковое или логическое выражение. Арифметическое выражение содержит четыре арифметические операции — сложение, вычитание, умножение и деление. Строковое выражение представлено одной строковой операцией конкатенации (||). Логическое выражение может содержать операцию отрицания (NOT), дизъюнкцию (логическое ИЛИ, ключевое слово OR) и конъюнкцию (логическое И, ключевое слово AND).

**Увеличение значения генератора**

Конструкция NEXT VALUE FOR <имя генератора> является аналогом вызова функции GEN_ID(<имя генератора>, 1). Значение указанного генератора увеличивается на единицу, и конструкция возвращает новое значение.

**Использование встроенных функций**

Существуют обычные встроенные в SQL функции и агрегатные функции в операторе SELECT.

Обычная встроенная функция — это функция, которой передается один или более параметров, функция не связана с оператором выборки данных SELECT. Такая функция возвращает ровно одно значение. Параметры передаются встроенным функциям на основании принятого для каждой функции синтаксиса. Описание встроенных функций см. в приложении E «Функции».

Агрегатные функции в операторе SELECT — функции, определенные в языке SQL СУБД Ред База Данных. Они работают не с одним фиксированным набором параметров, а с группой значений, полученных при выполнении оператора выборки данных SELECT из таблицы, хранимой процедуры выбора или из представления, описанного в базе данных. Агрегатные функции используются внутри списка выбора оператора SELECT. Синтаксис представлен в листинге 4.6.

**Листинг 4.6. Синтаксис агрегатной функции в операторе SELECT**

```
<агрегатная функция в операторе SELECT> ::=
   SELECT {
       COUNT ({[ALL | DISTINCT] <выражение> | *})
     | SUM ([ALL | DISTINCT] <выражение>)
     | AVG ([ALL | DISTINCT] <выражение>)
     | MAX ([ALL | DISTINCT] <выражение>)
     | MIN ([ALL | DISTINCT] <выражение>)
     | LIST ([ALL | DISTINCT] <выражение>) [, '<разделитель>']
     | CORR (<выражение1>, <выражение2>)
     | COVAR_POP (<выражение1>, <выражение2>)
     | COVAR_SAMP (<выражение1>, <выражение2>)
     | STDDEV_POP (<выражение>)
     | STDDEV_SAMP (<выражение>)
     | VAR_POP (<выражение>)
     | VAR_SAMP (<выражение>)
     | REGR_AVGX (y, x)
     | REGR_AVGY (y, x)
     | REGR_COUNT (y, x)
     | REGR_INTERCEPT (y, x)
     | REGR_R2 (y, x)
     | REGR_SLOPE (y, x)
     | REGR_SXX (y, x)
     | REGR_SXY (y, x)
     | REGR_SYY(y, x) }
   <предложение FROM>
   [<предложение WHERE>]
```

Оператор SELECT выбирает из указанной таблицы, хранимой процедуры или представления на основании заданного условия (если указано предложение WHERE) некоторое количество значений. Агрегатная функция внутри оператора SELECT выполняет соответствующие действия и возвращает одно значение.

Выражением может быть столбец таблицы, константа, переменная, неагрегатная функция или UDF. Агрегатные функции в качестве выражения не допускаются.

Функция COUNT подсчитывает количество указанных объектов. Задание параметра * означает, что функция подсчитывает все полученные оператором SELECT строки. Ключевое слово ALL указывает, что подсчитываются все значения, полученные оператором. Ключевое слово ALL предполагается по умолчанию. Этот вариант эквивалентен по результатам заданию параметра *. Ключевое слово DISTINCT задает подсчет только отличающихся значений. Одинаковыми считаются и значения, содержащие NULL.

Функция SUM возвращает сумму значений полученных строк. Здесь выражение может иметь числовой тип данных. Необязательное ключевое слово ALL указывает, что суммируются все значения, полученные оператором SELECT. Это вариант по умолчанию. Ключевое слово DISTINCT задает суммирование только отличающихся значений. При этом одинаковыми считаются и значения NULL.

Функция AVG возвращает среднее значение указанных значений полученных оператором SELECT строк. Здесь выражением также может быть значение, имеющего числовой тип данных. Ключевое слово ALL указывает, что в расчете среднего принимают участие все значения, полученные оператором SELECT (вариант по умолчанию). Ключевое слово DISTINCT позволяет включить в

расчет только отличающиеся значения.

Функции MAX и MIN задают, соответственно, выбор максимального и минимального значения среди всех значений, полученных в операторе SELECT строк. Ключевое слово ALL требует включения в процесс поиска всех значений, соответствующих условиям поиска, DISTINCT — только отличающихся; в этом случае вначале удаляются дубликаты значений, а затем в полученном списке отыскивается максимальное (минимальное) значение. Функции MAX и MIN применяются не только к числовым, но также и к строковым столбцам, к столбцам типа данных дата и время. Операции будут выполнены правильно, если для соответствующих строковых столбцов правильно заданы набор символов и порядок сортировки.

Функция LIST объединяет в один объект типа BLOB все данные, полученные из указанного выражения. Ключевое слово ALL (принимается по умолчанию) указывает, что в список попадают все значения, полученные оператором SELECT. Ключевое слово DISTINCT позволяет включить в список только отличающиеся значения. В функции также можно задать разделитель — 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. Здесь также можно задать два подряд идущих апострофа. В этом случае никакой разделитель не будет использован, значения будут «склеиваться» друг с другом.

Статистическая функция CORR возвращает коэффициент корреляции для пары выражений, возвращающих числовые значения.

Статистическая функция COVAR_POP возвращает ковариацию совокупности пар выражений с числовыми значениями.

Статистическая функция COVAR_SAMP возвращает выборочную ковариацию пары выражений с числовыми значениями.

Статистическая функция STDDEV_POP возвращает среднеквадратичное отклонение для группы.

Статистическая функция STDDEV_SAMP возвращает стандартное отклонение для группы.

Статистическая функция VAR_POP возвращает выборочную дисперсию для группы.

Статистическая функция VAR_SAMP возвращает несмещённую выборочную дисперсию для группы.

Функция линейной регрессии REGR_AVGX вычисляет среднее независимой переменной линии регрессии.

Функция линейной регрессии REGR_AVGY вычисляет среднее зависимой переменной линии регрессии.

Функция линейной регрессии REGR_COUNT возвращает количество непустых пар, используемых для создания линии регрессии.

Функция линейной регрессии REGR_INTERCEPT вычисляет точку пересечения линии регрессии с осью Y.

Функция линейной регрессии REGR_R2 вычисляет коэффициент детерминации, или R-квадрат, линии регрессии.

Функция линейной регрессии REGR_SLOPE вычисляет угол наклона линии регрессии.

Функции линейной регрессии REGR_SXX, REGR_SXY и REGR_SYY показывает диагностическую статистику, используемую для анализа регрессии.

Подробнее о каждой статистической функции и каждой функции линейной регрессии см. в Приложении Е.

**Функция UDF**

Значением в предложении CHECK также может быть обращение к функции, определенной пользователем (User Defined Function, UDF — см. приложение Г).

**Пустое значение**

В качестве значения может быть указано пустое значение NULL. Не следует NULL включать в какую-либо операцию сравнения. Результатом всегда будет неопределенное значение UNKNOWN. Лучше использовать конструкции IS NULL и IS NOT NULL.

# Предикаты сравнения

## Оператор IN

Оператор IN, синтакис которого выглядит следующим образом:

```
<значение> [NOT] IN ({<значение> [, <значение> ...] | <поиск одного>})
```

указывает, что значение, помещаемое в столбец, основанный на этом домене, оператором INSERT или изменяемое в столбце оператором UPDATE должно находиться (или не находиться, если указано ключевое слово NOT) в заданном списке.

Список должен заключаться в скобки, а значения разделяться запятыми. Символьные данные чувствительны к регистру. В списке допускается и пустое значение NULL. Любое значение в списке может быть представлено оператором SELECT, заключенным в круглые скобки, который возвращает одно значение. Весь список также можно задать и с использованием оператора SELECT, который выбирает произвольное количество значений ровно одного столбца из таблицы (таблиц), представления или хранимой процедуры.

Этот оператор может относиться к любому типу данных, кроме BLOB.

**Пример 1.** Если нам нужно смоделировать логический тип данных, то можно записать такое условие домена:

```
CHECK (VALUE IN ('0', '1'))
```

Здесь ведется проверка на вводимое значение, которое должно быть либо 0 (FALSE, «ложь»), либо 1 (TRUE, «истина»). Далее в этой главе будет приведен и более полный пример логического типа данных, включающего и неопределенное значение UNKNOWN.

> На самом деле в этом варианте неявно допускается и пустое значение. Чтобы в таком условии запретить использование значения NULL, следует явно записать:
>
> ```
> CHECK (VALUE IN ('0', '1') AND VALUE IS NOT NULL)
> ```

**Пример 2.** Если требуется создать домен, который в дальнейшем будет использоваться для столбцов таблиц, ссылающихся на таблицу стран COUNTRY, то для такого домена можно задать условие, чтобы помещаемые в столбец коды соответствовали одному из значений строки в таблице стран. Оператор SELECT в подобных конструкциях должен заключаться в круглые скобки.

```
CREATE DOMAIN D_SELECT AS CHAR(3)
  CHECK(VALUE IN (SELECT CODCOUNTRY FROM COUNTRY));
```

## Оператор BETWEEN

Оператор требует, чтобы значение находилось (или не находилось, если указано NOT) в заданном диапазоне, включая граничные значения. Для корректной обработки такого условия нужно, чтобы первое значение было меньше или равно второму.

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

Любое значение в этом операторе может быть представлено оператором SELECT, заключенным в круглые скобки, который возвращает одно значение.

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:

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

**Пример.** Примером использования оператора может служить проверка на то, что в односимвольное поле должны вводиться только десятичные цифры:

```
CHECK (VALUE BETWEEN '0' AND '9')
```

Если для такого столбца допускаются и пустые значения, то в условие домена нужно внести соответствующее добавление:

```
CHECK ((VALUE IS NULL) OR (VALUE BETWEEN '0' AND '9'))
```

Чтобы задать условие, что вводимое значение не должно содержать букв кириллицы в нижнем регистре, следует указать:

```
CHECK (VALUE NOT BETWEEN 'а' AND 'я')
```

Если же нужно задать условие, что значение не должно содержать букв кириллицы в любом регистре — ни строчных, ни прописных, — следует к ключевому слову `VALUE` применить функцию `UPPER`, переводящую все буквы во вводимом значении в прописные:

```
CHECK (UPPER(VALUE) NOT BETWEEN 'А' AND 'Я')
```

Можно также использовать и функцию `LOWER`, переводящую буквы в строчные:

```
CHECK (LOWER(VALUE) NOT BETWEEN 'а' AND 'я')
```

Результат будет таким же. Эти проверки будут выполняться совершенно корректно для букв кириллицы, если для домена будет указан порядок сортировки `PXW_CYRL`.

### Оператор `LIKE`

Этот оператор задает проверку наличия (или отсутствия в случае указания `NOT`) во вводимом значении символьного типа данных указанных символов.

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

Значением в этом операторе может быть и оператор `SELECT`, заключенный в круглые скобки и возвращающий одно значение.

Строка чувствительна к регистру. В исходной строке можно указать шаблонные символы `%` и `_`. Символ процента задает произвольное количество, в том числе и нулевое, любых символов. Шаблонный символ «знак подчеркивания» задает ровно один произвольный символ.

**Примеры.** Следующее условие требует, чтобы вводимое значение начиналось с символа «8», за которым может идти произвольное количество любых символов:

```
CHECK (VALUE LIKE '8%')
```

Для задания ввода телефонного номера можно указать, что номер должен начинаться с восьмерки, за которым в скобках идет трехсимвольный код города, а затем местный номер телефона, содержащий произвольное количество символов:

```
CHECK (VALUE LIKE '8(___)%')
```

Чтобы указать, что в номере телефона должен присутствовать и код страны, можно указать, например, такой оператор CHECK:

```
CHECK (VALUE LIKE '+%(___)%')
```

Код страны должен начинаться с символа «плюс» и может содержать несколько цифр, количество которых заранее неизвестно.

Чтобы задать в начале, в середине или в конце строки требование существования каких-либо символов, нужно перед этими символами и после них поставить символы %, например:

```
CHECK (VALUE LIKE '%мир%')
```

Здесь подстрока «мир» должна присутствовать в любом месте вводимой строки — в начале, в середине или в конце. Именно в указанном регистре.

Чтобы сделать эту конструкцию нечувствительной к регистру, следует также использовать функцию UPPER или LOWER:

```
CHECK (UPPER(VALUE) LIKE '%МИР%')
```

> Одним из источников ошибок может быть тот случай, когда при использовании функции UPPER соответствующий литерал в операторе задается строчными буквами. Например, если в предыдущем условии литерал записать в виде '%мир%', то такое условие никогда не будет выполнено.

Необязательное ключевое слово ESCAPE позволяет в строку поиска включить в виде отыскиваемых символов и сами шаблонные символы % и _. Здесь нужно указать символ, который должен предшествовать в строке поиска шаблонному символу, когда такой шаблонный символ должен рассматриваться сервером базы данных как обычный символ. Например, для ввода двухсимвольного числа, задающего проценты, можно использовать следующее условие:

```
CHECK (VALUE LIKE '__@%' ESCAPE '@')
```

Здесь символ «@» является тем символом, который отменяет использование шаблонного символа процент в строке поиска. Допустимыми будут значения 10%, 01%, однако попытка ввода строки 1% в данном случае вызовет ошибку. Чтобы не ограничивать размерность процентов, можно использовать следующее условие:

```
CHECK (VALUE LIKE '%@%' ESCAPE '@')
```

В этом случае можно будет вводить и 1000%.

Для того чтобы в строке можно было использовать обычным образом и символ, заданный в предложении ESCAPE, необходимо указать его в строке условия дважды. Например, при указании адреса электронной почты обычно используется символ @. Его в адресе e-mail следует повторить дважды:

```
CHECK (VALUE LIKE 'abcde@@red-soft.biz ...' ESCAPE '@')
```

В последней версии Ред База Данных работа сервера базы данных не завершается аварийно, даже если в ESCAPE задано пустое значение NULL.

### Оператор SIMILAR TO

Оператор SIMILAR TO проверяет соответствие вводимого значения с шаблоном регулярного выражения SQL.

```
<значение> [NOT] SIMILAR TO <значение> [ESCAPE '<символ>']
```

В отличие от некоторых других языков 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.

### Оператор IS NULL

В операторе VALUE IS [NOT] NULL проводится проверка вводимых данных на пустое значение. Оператор может вернуть только истинностное значение TRUE или FALSE, значение UNKNOWN невозможно. Такой оператор более естественно использовать в логическом выражении вместе с другими логическими условиями. Например, если в домене, моделирующем логический тип данных, нужно использовать не двухзначную логику (только «истина» и «ложь»), а трехзначную, используемую в SQL, введя понятие неопределенного значения, то для такого домена следует несколько усложнить условие:

```
CHECK ((VALUE IN ('0', '1')) OR (VALUE IS NULL))
```

Пустое значение будет тем самым третьим, неопределенным логическим значением (UNKNOWN), что хорошо согласуется с нормами, принятыми в SQL.

> На самом деле проверка на пустое значение в данном случае является излишней, поскольку вариант IN допускает и значение NULL. Однако с целью документирования скриптов создания объектов базы данных имеет смысл явно указать этот оператор.

Обратите внимание на наличие достаточно большого количества скобок, которые определяют порядок выполнения действий. Для сложных выражений во избежание двусмысленностей и ошибок рекомендуется всегда использовать скобки.

### Оператор IS DISTINCT FROM

Оператор выполняет проверку на равенство (неравенство, если задано NOT). В отличие от операторов равно (=) и не равно (!=) этот оператор трактует два пустых значения NULL как равные друг другу. Как и в случае оператора IS NULL данный оператор всегда возвращает либо TRUE, либо FALSE.

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

### Функции ALL, SOME, ANY

Синтаксис использования этих функций:

```
<значение> <оператор сравнения> {ALL | SOME | ANY} (<поиск одного>)
```

Здесь используется оператор сравнения. Аргументом любой из функций является оператор SELECT, возвращающий произвольное количество значений одного столбца таблицы, представления или хранимой процедуры выбора. Допустимо получение оператором SELECT также и пустого значения.

Функция ALL вернет значение «истина», если сравнение будет истинным для всех значений, полученных из оператора SELECT.

**Пример 1.** Следующий домен создается для столбца кода страны CODCOUNTRY в таблице организаций FIRM. В условии домена используется функция ALL, которая проверяет, что все организации расположены в одной стране:

```
CREATE DOMAIN D_ALL AS CHAR(3)
  CHECK (VALUE = ALL (SELECT CODCOUNTRY FROM FIRM));
```

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

**Пример 2.** Следующий домен создается для кода страны CODCOUNTRY в таблице организаций FIRM. Здесь выполняется проверка на то, чтобы код страны присутствовал хотя бы в одной строке таблицы стран:

```
CREATE DOMAIN D_ANY AS CHAR(3)
  CHECK (VALUE = ANY (SELECT CODCOUNTRY FROM COUNTRY));
```

Для сравнения см. следующий пример.

### Функция EXISTS

Синтаксис этой функции:

```
EXISTS (<поиск многих>)
```

Аргументом функции EXISTS является оператор SELECT, возвращающий произвольное количество любых столбцов таблицы, представления или хранимой процедуры выбора.

Результатом будет «истина», если оператор SELECT вернет хотя бы одно значение, соответствующее условиям поиска, заданным в предложении WHERE.

**Пример.** Следующий домен создается опять же для кода страны в таблице организаций FIRM. Здесь выполняется более естественная проверка на то, чтобы код страны присутствовал хотя бы в одной строке таблицы стран:

```
CREATE DOMAIN D_EXISTS AS CHAR(3)
  CHECK (EXISTS (SELECT CODCOUNTRY
                 FROM COUNTRY
                 WHERE CODCOUNTRY = VALUE));
```

### Функция SINGULAR

Синтаксис функции:

```
SINGULAR (<поиск многих>)
```

Аргументом функции SINGULAR является оператор SELECT, возвращающий произвольное количество любых столбцов таблицы, представления или хранимой процедуры.

Результатом будет «истина», если оператор SELECT вернет в точности одно значение, соответствующее условиям поиска, заданным в предложении WHERE.

```
CREATE DOMAIN D_SINGULAR AS CHAR(3)
  CHECK (SINGULAR(SELECT CODCOUNTRY
                  FROM COUNTRY
                  WHERE CODCOUNTRY = VALUE));
```

Использование функций ALL, ANY, SOME, EXISTS и SINGULAR не является естественным для доменов, поскольку в них используются операторы SELECT, обращающиеся к уже существую-

щим в базе данных таблицам. Обычно домены создаются до того, как будут сформированы таблицы, чтобы столбцы таблиц ссылались на созданные домены. Тем не менее, такие проверки работают и для доменов. Более рациональным является использование этих функций в условиях столбцов таблицы или в условиях всей таблицы, а также в операторе выборки данных SELECT. Подробности см. в главе 5 «Работа с таблицами».

### Оператор CONTAINING

Оператор задает проверку на присутствие во вводимом значении (или отсутствие в случае использования NOT) указанных символов. Символы могут располагаться в любом месте вводимой строки. Этот оператор нечувствителен к регистру. В операторе нельзя использовать шаблонные символы знак процента % и подчеркивания _. Эти символы будут восприниматься как обычные, а не шаблонные символы. Синтаксис оператора:

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

Результатом будет «истина», TRUE, если значение в левой части выражения будет содержать в качестве своей части значение, указанное в правой части.

Значением в этом операторе может быть и оператор SELECT, заключенный в круглые скобки и возвращающий одно значение или NULL.

**Пример.** При рассмотрении оператора LIKE приводился пример поиска подстроки «мир». Похожую проверку можно выполнить при использовании оператора CONTAINING, и этот вариант гораздо проще.

```
CHECK (VALUE CONTAINING 'мир')
```

В последнем случае отсутствует чувствительность к регистру. Условию будут удовлетворять и вводимые строки, содержащие такие символы, как «МИР», «Мир», «мИр» и т.д. Здесь нет необходимости применять встроенную функцию UPPER или LOWER.

### Оператор STARTING WITH

Оператор задает проверку на то, что вводимая строка начинается (или не начинается в случае задания NOT) с указанных символов. Оператор чувствителен к регистру, однако такое ограничение также можно обойти, применив функцию UPPER или LOWER.

Синтаксис этого оператора:

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

Значением в этом операторе может быть и оператор SELECT, заключенный в круглые скобки и возвращающий одно значение или NULL.

**Пример.** Чтобы указать, например, что вводимое значение не должно начинаться с буквы Q в любом регистре, нужно записать условие домена в следующем виде:

```
CHECK (UPPER(VALUE) NOT STARTING WITH 'Q')
```

## Логические операции с условиями домена

Для условия домена можно использовать более сложные логические конструкции, заключая отдельные условия в скобки, выполняя отрицание условия (используя ключевое слово NOT), выполняя логические операции конъюнкции (логическое И, ключевое слово AND) и дизъюнкции (логическое ИЛИ, ключевое слово OR).

В SQL используется не обычная двухзначная, а трехзначная логика. В ней присутствует три значения — TRUE (истина), FALSE (ложь) и UNKNOWN (неопределенное или неизвестное значение). Условие считается выполненным, если оно возвращает только значение TRUE. Следующие таблицы, называемые таблицами истинности, дают точное определение логическим операциям отрицания, дизъюнкции и конъюнкции в трехзначной логике.

В операции отрицания (NOT) присутствует один операнд. Результат выполнения отрицания в зависимости от значения операнда представлен в таблице 4.1.

Таблица 4.1 — Операция отрицания NOT

| Операнд | NOT операнд |
|---|---|
| TRUE | FALSE |
| FALSE | TRUE |
| UNKNOWN | UNKNOWN |

В операции дизъюнкции (логическое ИЛИ, OR) участвуют два операнда. Результат выполнения дизъюнкции двух операндов в зависимости от значений операндов показан в таблице 4.2.

Таблица 4.2 — Операция дизъюнкции OR

| Операнд 1 | Операнд 2 | Операнд 1 OR Операнд 2 |
|---|---|---|
| TRUE | TRUE | TRUE |
| TRUE | FALSE | TRUE |
| FALSE | TRUE | TRUE |
| FALSE | FALSE | FALSE |
| TRUE | UNKNOWN | TRUE |
| FALSE | UNKNOWN | UNKNOWN |
| UNKNOWN | TRUE | TRUE |
| UNKNOWN | FALSE | UNKNOWN |
| UNKNOWN | UNKNOWN | UNKNOWN |

В операции конъюнкции (логическое И, AND) участвуют два операнда. Результат выполнения конъюнкции в зависимости от значений операндов показан в таблице 4.3.

Таблица 4.3 — Операция конъюнкции AND

| Операнд 1 | Операнд 2 | Операнд 1 AND Операнд 2 |
|---|---|---|
| TRUE | TRUE | TRUE |
| TRUE | FALSE | FALSE |
| FALSE | TRUE | FALSE |
| FALSE | FALSE | FALSE |
| TRUE | UNKNOWN | UNKNOWN |
| FALSE | UNKNOWN | FALSE |
| UNKNOWN | TRUE | UNKNOWN |
| UNKNOWN | FALSE | FALSE |
| UNKNOWN | UNKNOWN | UNKNOWN |

Порядок выполнения операций в условии домена следующий:

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

Операции с одинаковым приоритетом выполняются слева направо.

> В случае использования домена, содержащего предложение `CHECK`, при создании столбца таблицы, где также указывается предложение `CHECK`, осуществляется конъюнкция обоих условий, то есть используется логическая операция И.

## 4.2 Изменение домена

Для изменения характеристик существующего в базе данных домена используется оператор `ALTER DOMAIN`. Синтаксис оператора показан в листинге 4.7.

**Листинг 4.7. Синтаксис оператора изменения домена `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>]
]];
```

В одном операторе `ALTER DOMAIN` можно выполнить любое количество изменений домена.

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.

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 пустое значение.

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.

Можно также поменять тип данных домена при помощи предложения `TYPE`. В предложении `CHARACTER SET` можно изменить набор символов и порядок сортировки (предложение `COLLATE`). Если в базе данных существуют таблицы, содержащие столбцы, основанные на данном домене, у которого

меняется тип данных, и такие таблицы уже содержат некоторые данные, то попытка изменения типа данных у домена может привести к ошибке этой операции в случае возникновения конфликта. В частности, при смене чувствительного к регистру ордера на нечувствительный могут быть ошибки уникальности. В этом случае изменение домена не произойдёт.

```
create domain d1 varchar(30) character set UTF8 collate UNICODE;
alter domain d1 type varchar(30) character set UTF8 collate UCS_BASIC;
```

Предложение `SET NOT NULL` устанавливает ограничение `NOT NULL` для домена. В этом случае для переменных и столбцах базирующихся на домене значение `NULL` не допускается. Предложение `DROP NOT NULL` удаляет ограничение `NOT NULL` для домена.

Изменить существующий домен может владелец домена (его создатель), пользователь с административными привилегиями или пользователь с привилегией `ALTER ANY DOMAIN`.

> Изменение характеристик домена в базе данных, где в созданных таблицах существуют столбцы, основанные на этом домене, может приводить к неприятным последствиям, включая потерю данных и невозможность использования существующих данных таких таблиц.

**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.

## 4.3 Удаление домена

Для удаления домена используется оператор `DROP DOMAIN`. Его синтаксис представлен в листинге 4.8.

**Листинг 4.8. Синтаксис оператора удаления домена `DROP DOMAIN`**

```
DROP DOMAIN <имя домена>;
```

Нельзя удалить домен, на который ссылаются столбцы существующих таблиц базы данных или внутренние переменные хранимых процедур и триггеров. Предварительно нужно удалить все столбцы и переменные, ссылающиеся на этот домен. Удаление доменов для заполненной данными базы данных не является хорошей практикой.

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.

## 4.4 Примечание домена

Для существующего домена вы можете создать комментарий, используя оператор `COMMENT ON DOMAIN` следующего вида (см. листинг 4.9):

**Листинг 4.9. 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 5

# Работа с таблицами

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 может содержать не менее одного столбца 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.

## 5.1 Создание таблиц

Таблица создается оператором `CREATE TABLE`. Синтаксис оператора представлен в листинге 5.1.

**Листинг 5.1. Синтаксис оператора создания таблицы** `CREATE TABLE`

```
CREATE TABLE <имя таблицы>
   [EXTERNAL [FILE] '<спецификация файла>' [ADAPTER 'CSV']]
   (<определение столбца> [, {<определение столбца> | <ограничение таблицы>}...])
   [SQL SECURITY {DEFINER | INVOKER}];
```

Имя таблицы должно быть уникальным среди имен таблиц базы данных, хранимых процедур и представлений, описанных в этой базе данных.

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.

Необязательное предложение `SQL SECURITY {DEFINER | INVOKER}` определяет, в контексте какого пользователя будет проходить работа с таблицей. Ключевое слово `INVOKER` (значение по умолчанию) указывает, что таблица вызывается с правами текущего пользователя. Задание ключевого слова `DEFINER` означает, что таблица вызывается 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`.

Создать новую таблицу может администратор и пользователь с привилегией `CREATE TABLE`. Пользователь, создавший таблицу, становится её владельцем.

### 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 5.2. 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.

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 );
```

## 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.

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(24),
    message CHAR(100),
```

```
    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(24));
    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:

```
2015-10-07 15:19:03.4110Shall I compare thee to a summer's day?
2015-10-07 15:19:58.7600Thou 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"
23/06/1954,Disneyland,"I'm going to ""Walt Disney World""!"
15/07/1934,,JUST DO IT,aaa,bbb
,
```

Пример использования CSV адаптера:

```
create table CSV_EXT external 'D:\externals\table.csv' adapter 'CSV'(
    Sdate DATE,
    Company VARCHAR(30),
    Slogan VARCHAR(100)
)
```

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>
```

# Определение столбца

## Синтаксис определения столбца

Синтаксис описания столбца таблицы представлен в листинге 5.3.

**Листинг 5.3. Синтаксис определения столбца таблицы**

```
<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 BY DEFAULT AS IDENTITY [(START WITH <стартовое значение>)]
   [<ограничение столбца>]
```

Столбец должен иметь имя, уникальное только в данной таблице. В других таблицах могут присутствовать столбцы с тем же именем.

Для столбца должен быть задан либо тип данных, либо имя домена, характеристики которого будут скопированы в этот столбец, либо должно быть указано, что столбец является вычисляемым (`COMPUTED BY` или семантически эквивалентная конструкция `GENERATED ALWAYS AS`).

### Задание типа данных

Подробное описание типов данных, допустимых операций преобразования и других встроенных в SQL функций работы с данными см. в главе 2 «Типы данных Ред База Данных» и в приложении Е «Функции». Синтаксис задания типа данных столбца таблицы показан в листинге 5.4.

**Листинг 5.4. Синтаксис задания типа данных столбца таблицы**

```
<тип данных> ::= {
    {SMALLINT | INTEGER | BIGINT} [<размерность массива>]
  | BOOLEAN [<размерность массива>]
  | {FLOAT | DOUBLE PRECISION} [<размерность массива>]
  | {DATE | TIME | TIMESTAMP} [<размерность массива>]
  | {DECIMAL | NUMERIC} [(<точность> [ ,<масштаб>])] [<размерность массива>]
  | {CHAR | CHARACTER | CHARACTER VARYING | VARCHAR} [(<размер>)]
      [CHARACTER SET <набор символов>] [<размерность массива>]
  | {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 и столбца с типом данных BLOB подтипа TEXT можно в предложении CHARACTER SET указать набор символов, если требуется набор, отличный от набора символов по умолчанию, установленного для всей базы данных. Кроме того, в предложении COLLATE можно задать и порядок сортировки (для типа данных BLOB использование COLLATE недопустимо).

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.

Если для столбца с типом данных BLOB указан подтип, то для такого столбца нельзя задавать набор символов. Считается, что он предопределен подтипом.

### 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 при создании столбца таблицы осуществляется ссылка на домен, содержащий предложение CHECK, а при описании столбца также задается ограничение CHECK, то в результате для столбца будет сформировано условие, являющееся конъюнкцией (логическое И) обоих условий.

> Если при описании столбца таблицы не задается базовый домен, а указывается тип данных, то для такого столбца система все равно создаст системный домен с уникальным именем, поместив в него все описанные для столбца характеристики.

## 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 8 «Операторы DML»). 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 2 «Типы данных Ред База Данных»). 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.

## 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.** В справочной таблице стран в нашем примере код страны является первичным ключом. По этой причине он объявлен с предложением NOT NULL:

```
CODCOUNTRY CHAR(3) NOT NULL, /* Код страны */
...
CONSTRAINT PK_COUNTRY PRIMARY KEY (CODCOUNTRY), ...
```

## Ограничение столбца

Ограничение столбца (его иначе еще называют ограничением на уровне столбца) записывается сразу после определения других характеристик столбца. Существует четыре вида ограничений столбца: первичный ключ (ключевые слова PRIMARY KEY), уникальный ключ (UNIQUE), внешний ключ (REFERENCES) и ограничение значения столбца, CHECK. Синтаксис ограничения столбца представлен в листинге 5.5.

**Листинг 5.5. Синтаксис задания ограничения столбца**

```
<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>)
  }

<предложение USING> ::= USING [ASC[ENDING] | DESC[ENDING]] INDEX <имя индекса>
```

Необязательное предложение CONSTRAINT задает имя ограничения столбца. Если имя не указано, система присваивает ограничению системное имя, например, INTEG_28. 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.

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.

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 7 «Работа с индексами»). Такое поведение может привести к аварийному завершению работы сервера базы данных при выборке данных на основании каких-либо условий, включающих использование уникального ключа.

**Ограничение PRIMARY KEY**

Ограничение PRIMARY KEY определяет первичный ключ. В отличие от уникального ключа в таблице может быть только один первичный ключ.

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.

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.

Нельзя в базе данных создавать индекс, по структуре соответствующий первичному ключу (см. главу 7 «Работа с индексами»). 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.

Первичный ключ может состоять из одного или более столбцов таблицы. Для каждого столбца, входящего в состав первичного ключа, необходимо при его описании явно задать характеристику NOT NULL, что запрещает помещать в эти столбцы пустые значения. В этом его отличие от уникального ключа. Столбцы уникального ключа могут получать пустое значение NULL.

Основным свойством первичного ключа является его уникальность — в таблице не может присутствовать двух различных строк с одним и тем же значением столбцов, входящих в состав первичного ключа. По значению первичного ключа можно отыскать конкретную строку в таблице или определить, что соответствующая строка отсутствует.

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 База 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 REFERENCES**

Ограничение REFERENCES определяет внешний ключ. 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.

Первичный или уникальный ключ часто называют родительскими ключами.

В предложении REFERENCES указывается имя таблицы (главной, родительской), на первичный/уникальный ключ которой ссылается внешний ключ подчиненной, дочерней, таблицы, и имя первичного/уникального ключа в главной, родительской, таблице, на которую 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.

В этой конструкции предложение ON DELETE определяет, что произойдет с записями подчиненной, дочерней, таблицы при удалении соответствующей строки главной, родительской, таблицы:

- NO ACTION — не будет выполнено никаких действий. Обеспечение соответствия внешнего ключа первичному (уникальному) ключу должна выполнить сама клиентская программа либо для этого следует написать выполняемую хранимую процедуру, к которой должно осуществляться в процессе удаления строки обращение из клиентской программы, или специально созданный пользовательский триггер до удаления (BEFORE DELETE), выполняющий все необходимые установки значений внешнего ключа дочерней таблицы;
- CASCADE — в дочерней таблице должны быть автоматически удалены все записи, имеющие те же значения внешнего ключа, что и значение первичного (уникального) родительского ключа удаляемой 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.

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.

Как правило, на практике используются варианты CASCADE или, реже, SET NULL.

Предложение ON UPDATE определяет, что произойдет с записями подчиненной, дочерней, таблицы при изменении значения первичного/уникального ключа в строке главной, родительской, таблицы:

- NO ACTION — не будет выполнено никаких действий (значение по умолчанию). Обеспечение соответствия связи внешнего ключа / первичного (уникального) ключа должна выполнить сама клиентская программа или специально созданный для этого пользовательский триггер до изменения (BEFORE UPDATE) или выполняемая хранимая процедура, к которой в клиентской программе должно выполняться обращение сразу после изменения ключевых реквизитов родительской таблицы;
- CASCADE — в дочерней таблице должны быть автоматически изменены все значения внешнего ключа, имеющие те же значения, что и значение первичного (уникального) ключа изменяемой строки родительской таблицы. Это наиболее надежный вариант поведения, который не приводит к нарушению целостности данных. 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 для этого столбца. Может привести к нарушению ссылочной целостности данных при задании значения по умолчанию, не имеющему соответствия в родительской таблице. Если значение по умолчанию для столбца внешнего ключа не задано, то столбцу присваивается значение NULL, это позволяет избежать каких бы то ни было неприятностей при изменении родительского ключа;
- SET NULL — значения внешнего ключа всех соответствующих строк в дочерней таблице устанавливаются в пустое значение NULL. Также является хорошим вариантом поддержания ссылочной целостности данных, хотя и не приводит к нормальному поддержанию соответствия главная/подчиненная.

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.

На практике обычно используется вариант CASCADE, что в большинстве случаев является лучшим практическим решением.

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. Если в описании внешнего ключа было указано имя ограничения в предложении CONSTRAINT, то это имя будет присвоено автоматически создаваемому индексу. Рекомендуется каждому такому ограничению явно присваивать имя.

Предложение USING позволяет задать иное имя индекса для поддержания ограничения внешнего ключа.

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.

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 5.6.

**Листинг 5.6. 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>)
  | EXISTS (<поиск многих>)
  | 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>
}
```

**Оператор сравнения**

Оператором в этом условии столбца является оператор сравнения.

**Листинг 5.7. Синтаксис оператора сравнения для столбца таблицы**

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

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

| **Операторы** | = | <> , != , ^= | > | < | >= , !< , ^< | <= , !> , ^> |
|---|---|---|---|---|---|---|
| **Значение** | Равно | Не равно | Больше | Меньше | Больше или равно, не меньше | Меньше или равно, не больше |

Результатом сравнения, когда один из операндов или оба имеют значение NULL, 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 таком 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 целочисленным данным, с числом с фиксированной или с плавающей точкой.

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

Нельзя дату сравнивать со временем.

### Значение в условии столбца таблицы

Термин «значение» в синтаксисе условия столбца определяется следующим образом (см. листинг 5.8):

**Листинг 5.8. Синтаксис значения в условии столбца таблицы**

```
<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 }
```

Здесь можно указать имя столбца таблицы. Если столбец является массивом, то в квадратных скобках нужно указать и конкретный элемент массива. Если это одномерный массив, то указывается номер этого элемента (с учетом заданного диапазона значений для элементов массива). Для многомерных массивов нужно указать номера позиций каждого из диапазонов, разделяя их запятыми.

Литерал — это числовая константа, строковая константа, заключенная в апострофы, литерал даты или времени, предварительно определенный литерал, контекстная переменная.

Подробные описания характеристик и порядка использования предварительно определенных литералов и контекстных переменных см. в главе 2 «Типы данных Ред База Данных». Полные описания находятся в приложении Ж «Контекстные переменные».

### Использование встроенных функций

В SQL Ред База Данных существует два типа встроенных функций — обычные встроенные

функции и агрегатные функции в операторе SELECT.

Обычная встроенная функция — это функция, получающая один или более параметров, которая не связана с оператором выборки данных SELECT. Функция возвращает ровно одно значение. Параметры передаются таким функциям на основании принятого для каждой функции синтаксиса. Описание встроенных функций см. в приложении Е «Функции».

Агрегатные функции в операторе SELECT — функции, определенные в языке SQL Ред База Данных. Они работают не с одним фиксированным набором параметров, а с группой значений, полученных при выполнении оператора выборки данных SELECT из таблицы базы данных. Агрегатные функции используются внутри списка выбора оператора SELECT. Оператор SELECT выбирает из указанной таблицы на основании заданного условия некоторое количество значений. Агрегатная функция внутри оператора SELECT выполняет соответствующие расчеты и возвращает одно число.

- Функция COUNT подсчитывает количество указанных объектов;
- Функция SUM возвращает сумму указанных значений полученных строк;
- Функция AVG возвращает среднее арифметическое значение указанных значений полученных строк;
- Функции MAX и MIN задают, соответственно, поиск максимального и минимального значения среди всех значений полученных строк. Функции могут работать с любыми типами данных кроме BLOB;
- Функция LIST объединяет в один объект типа BLOB все данные, полученные из указанного выражения;
- Функции CORR, COVAR_POP, COVAR_SAMP, STDDEV_POP, STDDEV_SAMP, VAR_POP, VAR_SAMP используются для получения статистических показателей;
- Функции линейной регрессии REGR_AVGX, REGR_AVGY, REGR_COUNT, REGR_INTERCEPT, REGR_R2, REGR_SLOPE, REGR_SXX, REGR_SXY и REGR_SYY 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.

Подробные описания агрегатных функций см. в главе 4 «Работа с доменами». Синтаксис агрегатных функций описан в приложении Е «Функции».

Значением также может быть обращение к функции, определенной пользователем (User Defined Function, UDF — см. приложение Г «Функции, определенные пользователем (UDF)»).

В качестве значения может быть указано пустое значение NULL. Не следует NULL включать в какую-либо операцию сравнения. Результатом всегда будет неопределенное значение. Лучше использовать конструкции IS NULL и IS NOT NULL (см. ниже).

### Операторы SELECT, используемые в условии столбца

*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.

### Оператор IN

Оператор IN определяет, что вводимое в столбец значение должно находиться (или не находиться, если указано ключевое слово NOT) в заданном списке.

Список может быть представлен в виде явно указанных значений. Весь список заключается в круглые скобки. Отдельные значения должны разделяться запятыми. Любое значение в списке может быть представлено оператором SELECT, заключенным в круглые скобки, который возвращает

одно значение. Весь список также можно задать и с использованием оператора SELECT, который выбирает произвольное количество значений ровно одного столбца из таблицы (таблиц), представления или хранимой процедуры выбора.

Символьные данные чувствительны к регистру.

```
<значение> [NOT] IN ({<значение> [, <значение> ...]| <поиск одного>})
```

Этот оператор может относиться к любому типу данных, кроме BLOB.

Если нужно отключить чувствительность к регистру, то следует к значению в левой части этого выражения применить функцию UPPER, а значения в списке записывать прописными буквами. Можно также использовать функцию LOWER, записывая значения в списке строчными буквами.

**Пример.** Приведем пример использования оператора SELECT в качестве списка значений.

Пусть имеется таблица, содержащая списки регионов различных стран. Для России регион — это республика, край, область. Таблица имеет следующую структуру:

```
CREATE TABLE REGION (
   CODCOUNTRY CHAR(3) NOT NULL, /* Код страны */
   CODREGION CHAR(2) NOT NULL, /* Код региона */
   ...
);
```

Есть также таблица, содержащая сведения об организациях, FIRM. В этой таблице присутствует столбец CODREGION, задающий код региона, где располагается организация. Для этого столбца можно задать проверку на то, что код региона должен присутствовать среди записей таблицы регионов.

```
CODREGION CHAR(2)
   CHECK (CODREGION IN (SELECT CODREGION
                        FROM REGION
                        WHERE CODCOUNTRY = 'RUS'))
```

Здесь в качестве списка значений в операторе IN выбираются только коды регионов одной страны с кодом RUS — России.

### Оператор BETWEEN

В операторе проверяется, присутствует ли значение, записанное в левой части условия, в конкретном диапазоне значений, заданных в правой части условия, включая граничные значения. Начальное значение должно быть не больше конечного значения в диапазоне.

Любое значение в этом операторе может быть представлено оператором SELECT, заключенным в круглые скобки, который возвращает одно значение.

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

Условие будет истинным, если значение присутствует в указанном диапазоне при отсутствии ключевого слова NOT. При наличии ключевого слова NOT условие будет истинным, если значение отсутствует в указанном диапазоне.

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:

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

### Оператор LIKE

Этот оператор задает проверку наличия (или отсутствия в случае указания NOT) во вводимом значении символьного типа данных определенных символов.

Значением в этом операторе может быть и оператор SELECT, заключенный в круглые скобки и возвращающий одно значение.

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

Вариант является чувствительным к регистру. В исходной строке можно указать шаблонные символы % и _. Символ процента задает произвольное количество, в том числе и нулевое, любых символов. Знак подчеркивания задает ровно один произвольный символ.

Чтобы этот вариант можно было применять как к строчным, так и к прописным буквам, следует использовать функцию перевода букв в прописные UPPER или функцию перевода прописных букв в строчные LOWER.

```
UPPER(<значение>) LIKE '<строка, состоящая из прописных букв>'
```

Другой вариант:

```
LOWER(<значение>) LIKE '<строка, состоящая из строчных букв>'
```

Необязательное ключевое слово ESCAPE позволяет в строку поиска включить и сами шаблонные символы % и _. Здесь нужно указать символ, который должен предшествовать в строке поиска шаблонному символу, когда такой шаблонный символ должен рассматриваться как обычный символ, присутствующий в строке.

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

**Оператор SIMILAR TO**

Оператор SIMILAR TO проверяет соответствие вводимого значения с шаблоном регулярного выражения SQL.

```
<значение> [NOT] SIMILAR TO <значение> [ESCAPE '<символ>']
```

В отличие от некоторых других языков 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.

**Оператор IS NULL**

Этот оператор осуществляет проверку на пустое значение (или отсутствие пустого значения).

```
<значение> IS [NOT] NULL
```

Оператор может возвращать только два значения — TRUE или FALSE, значение UNKNOWN невозможно.

**Оператор IS DISTINCT FROM**

Оператор выполняет проверку на неравенство (равенство, если задано NOT). 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, но не UNKNOWN.

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

**Функции ALL, SOME, ANY**

Синтаксис использования этих функций:

```
<значение> <оператор сравнения> {ALL | SOME | ANY} (<поиск одного>)
```

Здесь может использоваться любой оператор сравнения, а также оператор IS [NOT] DISTINCT FROM. Аргументом любой из функций является оператор SELECT, возвращающий произвольное количество значений одного столбца таблицы, представления или хранимой процедуры выбора. Допустимо также получение оператором и пустого значения.

Функция ALL вернет значение «истина», если сравнение будет истинным для всех значений, полученных при выполнении оператора SELECT.

Ключевые слова SOME и ANY являются синонимами. Результатом будет «истина», если сравнение истинно хотя бы для одного значения, полученного при выполнении оператора SELECT.

**Пример.** При помещении новой строки в таблицу организаций FIRM для столбца, который будет ссылаться на код страны в таблице стран можно задать следующую проверку:

```
CHECK(CODCOUNTRY = ANY (SELECT CODCOUNTRY FROM COUNTRY))
```

### Функция EXISTS

Синтаксис этой функции:

```
EXISTS (<поиск многих>)
```

Аргументом функции EXISTS является оператор SELECT, возвращающий произвольное количество любых столбцов заданной таблицы.

Результатом будет «истина», если оператор SELECT вернет хотя бы одно значение, соответствующее условиям поиска, заданным в предложении WHERE.

**Пример.** При помещении новой строки в таблицу организаций FIRM для столбца, который будет ссылаться на код страны в таблице стран можно задать следующую проверку (см. предыдущий пример):

```
CHECK( EXISTS(SELECT CODCOUNTRY
              FROM COUNTRY
              WHERE COUNTRY.CODCOUNTRY = FIRM.CODCOUNTRY) )
```

### Функция SINGULAR

Синтаксис функции:

```
SINGULAR (<поиск многих>)
```

Аргументом функции SINGULAR является оператор SELECT, возвращающий произвольное количество любых столбцов заданной таблицы.

Результатом будет «истина», TRUE, если оператор SELECT вернет только одно значение, соответствующее условиям поиска, заданным в предложении WHERE.

### Оператор CONTAINING

Синтаксис оператора:

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

Результатом будет «истина», TRUE, если значение в левой части выражения будет содержать в качестве своей части значение, указанное в правой части. Этот оператор не чувствителен к регистру.

Значением в этом операторе может быть и оператор SELECT, заключенный в круглые скобки и возвращающий одно значение или NULL.

**Пример.** Если требуется, чтобы строковый столбец содержал в любом месте своего значения слово «дом», можно задать такую проверку:

```
CHECK( STRING_FIELD CONTAINING 'дом' )
```

### Оператор STARTING WITH

Синтаксис этого оператора:

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

Результатом будет «истина», TRUE, если значение в левой части выражения будет начинаться с символов, указанных в правой части. Оператор чувствителен к регистру, однако это ограничение также можно обойти, используя функцию UPPER или функцию LOWER.

Значением в этом операторе может быть оператор SELECT, заключенный в круглые скобки и возвращающий одно значение или NULL.

**Пример.** Чтобы указать, что вводимое значение не должно начинаться с буквы Q в любом регистре, нужно записать условие столбца в одном из следующих вариантов:

```
CHECK (UPPER(STRING_FIELD) NOT STARTING WITH 'Q')
```

или

```
CHECK (LOWER(STRING_FIELD) NOT STARTING WITH 'q')
```

### Логические операции с условиями столбца

При создании условия столбца можно использовать круглые скобки, чтобы задать высший приоритет выполнения части условия. Можно использовать логические операции отрицания (NOT), дизъюнкции (логическое ИЛИ, OR) и конъюнкции (логическое И, AND).

В SQL используется не обычная двухзначная, а трехзначная логика. В ней присутствует три значения — TRUE (истина), FALSE (ложь) и UNKNOWN (неопределенное или неизвестное значение). Следующие таблицы, называемые таблицами истинности, дают точное определение логическим операциям отрицания, дизъюнкции и конъюнкции.

В операции отрицания присутствует один операнд. Результат выполнения отрицания в зависимости от значения операнда представлен в таблице 13.3.

Таблица 5.1 — Операция отрицания NOT

| Операнд | NOT операнд |
|---|---|
| TRUE | FALSE |
| FALSE | TRUE |
| UNKNOWN | UNKNOWN |

В операции дизъюнкции (логическое ИЛИ) участвуют два операнда. Результат выполнения дизъюнкции в зависимости от значений операндов показан в таблице 5.2.

Таблица 5.2 — Операция дизъюнкции OR

| Операнд 1 | Операнд 2 | Операнд 1 OR Операнд 2 |
|---|---|---|
| TRUE | TRUE | TRUE |
| TRUE | FALSE | TRUE |
| FALSE | TRUE | TRUE |
| FALSE | FALSE | FALSE |
| TRUE | UNKNOWN | TRUE |
| FALSE | UNKNOWN | UNKNOWN |
| UNKNOWN | TRUE | TRUE |

| Операнд 1 | Операнд 2 | Операнд 1 OR Операнд 2 |
|---|---|---|
| UNKNOWN | FALSE | UNKNOWN |
| UNKNOWN | UNKNOWN | UNKNOWN |

В операции конъюнкции (логическое И) участвуют два операнда. Результат выполнения конъюнкции в зависимости от значений операндов показан в таблице 5.3.

Таблица 5.3 — Операция конъюнкции AND

| Операнд 1 | Операнд 2 | Операнд 1 AND Операнд 2 |
|---|---|---|
| TRUE | TRUE | TRUE |
| TRUE | FALSE | FALSE |
| FALSE | TRUE | FALSE |
| FALSE | FALSE | FALSE |
| TRUE | UNKNOWN | UNKNOWN |
| FALSE | UNKNOWN | FALSE |
| UNKNOWN | TRUE | UNKNOWN |
| UNKNOWN | FALSE | FALSE |
| UNKNOWN | UNKNOWN | UNKNOWN |

Порядок выполнения операций следующий:

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

Операции с одинаковым приоритетом выполняются слева направо.

**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 5.2 Изменение таблиц.

## Вычисляемые столбцы

Вычисляемый столбец задается предложением:

```
<имя столбца> [<тип данных>] {COMPUTED [BY] | GENERATED ALWAYS AS} (<выражение>)
```

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 Г). 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 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)
);
```

Другая таблица с двумя вычисляемыми столбцами, описывающая различные организации, содержит код страны, 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 главе 8 «Операторы DML», раздел «SELECT».

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 главе 8 «Операторы DML», раздел «SELECT». Описание полномочий пользователя к таблицам, процедурам, триггерам и представлениям см. в документе «Руководство администратора».

### Столбцы идентификации

Столбцы идентификации могут быть определены с помощью предложения GENERATED BY DEFAULT AS IDENTITY. Столбец идентификации представляет собой столбец, связанный с внутренним генератором последовательностей. Его значение устанавливается автоматически каждый раз, когда оно не указано в операторе INSERT. Необязательное предложение START WITH позволяет указать начальное значение отличное от нуля. 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.

## Определение ограничений таблицы

Ограничения таблицы являются более универсальным, удобным и наглядным способом описания ограничений таблицы, чем ограничения столбца. Они применяются не только к одному столбцу, но и к группе столбцов создаваемой (изменяемой) таблицы.

Ограничение таблицы описывается следующим синтаксисом (см. листинг 5.9):

**Листинг 5.9. Синтаксис задания ограничения таблицы**

```
<ограничение таблицы> ::=
  [CONSTRAINT <имя ограничения>]
  {
    PRIMARY KEY (<имя столбца> [, <имя столбца> ...]) [<предложение USING>]
  | UNIQUE (<имя столбца> [, <имя столбца> ...]) [<предложение USING>]
  | FOREIGN KEY (<имя столбца> [, <имя столбца> ...])
    REFERENCES <имя таблицы> (<имя столбца> [, <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>)
  }

<предложение USING> ::= USING [ASC[ENDING] | DESC[ENDING]] INDEX <имя индекса>
```

Ограничение таблицы, в отличие от ограничения столбца, может относиться как к одному отдельному столбцу, так и к группе столбцов этой таблицы. Столбцы, задаваемые в ограничении, должны быть уже описаны при создании или изменении таблицы. По этой причине ограничения

таблицы обычно размещаются в самом конце описания таблицы или впоследствии добавляются к описанию таблицы при использовании оператора ALTER TABLE.

Ограничению таблицы также можно присвоить имя, используя предложение CONSTRAINT. Имя ограничения должно быть уникальным среди имен всех ограничений всех таблиц, ограничений столбцов таблиц и имен всех индексов базы данных. Если не указано предложение USING, то соответствующий данному ограничению индекс получит имя создаваемого ограничения. Если же не задано также и имя ограничения, то этому индексу будет присвоено системное имя.

В случае задания предложения USING при описании первичного, уникального или внешнего ключа можно указать имя создаваемого индекса, поддерживающего соответствующее ограничение, а также и его упорядоченность — по возрастанию (ASCENDING — принимается по умолчанию) или по убыванию (DESCENDING).

## Ограничение первичного ключа

Предложение PRIMARY KEY задает ограничение первичного ключа. В состав первичного ключа в ограничении таблицы может входить один или более столбцов данной таблицы. Синтаксис задания ограничения первичного ключа:

```
PRIMARY KEY (<имя столбца> [, <имя столбца> ...]) [<предложение USING>]
```

Имена столбцов перечисляются в круглых скобках и отделяются друг от друга запятыми. Ни одни столбец, входящий в состав первичного ключа, не может иметь пустого значения. Каждый столбец первичного ключа должен быть явно описан с указанием атрибута NOT NULL. Таблица может иметь не более одного первичного ключа.

Для первичного ключа система автоматически создает индекс в момент создания таблицы или при добавлении в существующую таблицу ограничения первичного ключа в операторе ALTER TABLE. Если ограничению первичного ключа было назначено имя в предложении CONSTRAINT (при отсутствии предложения USING), то это имя присваивается создаваемому индексу. Если же было задано и предложение USING, то индекс для этого первичного ключа получает имя, указанное в предложении USING. Иначе индекс получает системное имя. В предложении USING можно также указать упорядоченность создаваемого индекса — по возрастанию значений индекса (ASCENDING) или по убыванию значений первичного ключа (DESCENDING). По умолчанию предполагается ASCENDING.

## Ограничение уникального ключа

Предложение UNIQUE задает ограничение уникального ключа. В состав уникального ключа в ограничении таблицы может входить произвольное количество столбцов данной таблицы.

```
UNIQUE (<имя столбца> [, <имя столбца>...]) [<предложение USING>]
```

Имена столбцов, входящих в состав уникального ключа, перечисляются в круглых скобках и отделяются друг от друга запятыми. В отличие от первичного ключа отдельные столбцы уникального ключа (не обязательно все, а только некоторые из них) могут иметь пустое значение NULL. При этом комбинация значений столбцов, входящих в состав уникального ключа, должна оставаться уникальной, исходя из того, что в подобной ситуации два пустых значения NULL считаются одинаковыми. Исключением является лишь случай, когда все столбцы уникального ключа имеют пустое значение. Таких строк с полностью «пустым» значением уникального ключа в одной таблице может быть произвольное количество.

Пусть, например, уникальный ключ таблицы состоит из двух столбцов K1 и K2, и для них не указано предложение NOT NULL. Тогда присутствие в такой таблице следующих строк является допустимым:

```
K1     K2
=====  =====
1      1
1      2
NULL   NULL
NULL   NULL
1      NULL
NULL   2
NULL   NULL
```

Однако попытка записать в эту таблицу еще и любую из следующих строк приведет к нарушению уникальности значения ключа:

```
K1     K2
=====  =====
1      NULL
NULL   2
```

Эти значения уникального ключа уже присутствуют в таблице. Такие действия вызовут исключение базы данных, строки в таблицу помещены не будут.

Таблица может иметь произвольное количество уникальных ключей.

Для уникального ключа система автоматически создает индекс. Если ограничению было назначено имя в предложении `CONSTRAINT`, то это имя присваивается созданному индексу. Если же было задано и предложение `USING`, то индекс для уникального ключа получает имя, указанное в этом предложении. Иначе индекс для уникального ключа получает системное имя. В предложении `USING` можно также указать и упорядоченность создаваемого индекса — по возрастанию (`ASCENDING` — значение по умолчанию) или по убыванию значений реквизитов уникального ключа (`DESCENDING`).

## Ограничение внешнего ключа

Предложение `FOREIGN KEY` задает ограничение внешнего ключа. Синтаксис этого ограничения на уровне таблицы несколько отличается от синтаксиса определения внешнего ключа на уровне столбца.

```
FOREIGN KEY (<имя столбца> [, <имя столбца> ...])
REFERENCES <имя таблицы> (<имя столбца> [,<имя столбца>...])[<предложение USING>]
   [ON DELETE { NO ACTION | CASCADE | SET DEFAULT | SET NULL }]
   [ON UPDATE { NO ACTION | CASCADE | SET DEFAULT | SET NULL }]
```

После ключевых слов `FOREIGN KEY` в скобках указывается список столбцов создаваемой (изменяемой) таблицы, которые включены в состав внешнего ключа. Может быть задан и один единственный столбец. Все столбцы, входящие в состав внешнего ключа, должны быть описаны в таблице ранее.

После ключевого слова `REFERENCES` указывается имя родительской таблицы, на первичный или уникальный ключ которой ссылается описываемый внешний ключ. Список имен столбцов родительской таблицы, входящих в состав первичного (уникального) ключа помещается сразу после имени таблицы в этом предложении и заключается в круглые скобки. Сама родительская таблица уже должна быть создана в базе данных. Если происходит ссылка именно на первичный ключ родительской таблицы, то список столбцов, включенных в состав первичного ключа этой таблицы, можно не указывать.

Для внешнего ключа система автоматически создает индекс. Если ограничению было назначено имя в предложении `CONSTRAINT` (при отсутствии предложения `USING`), то это имя присваивается созданному индексу. Если же было задано и предложение `USING`, то индекс для внешнего ключа получает имя, указанное в этом предложении. Иначе индекс получает системное имя. В предложении

USING можно также указать и упорядоченность автоматически создаваемого индекса — по возрастанию (ASCENDING — вариант, принимаемый по умолчанию) или по убыванию значений реквизитов внешнего ключа (DESCENDING).

> Структура внешнего ключа дочерней таблицы по количеству столбцов и по типам данных, включая размерность символьных данных, должна полностью соответствовать структуре первичного (уникального) ключа родительской таблицы, на который ссылается данный внешний ключ. Совпадения имен не требуется.

В SQL в некоторых случаях не требуется совпадение размеров символьных столбцов, см., например, предложение UNION в операторе SELECT (глава 8 «Операторы DML», раздел «SELECT»), однако, для полного исключения ошибочных ситуаций такое соответствие желательно. Упорядоченность индекса внешнего ключа должна в точности соответствовать упорядоченности индекса первичного (уникального) ключа, на который ссылается внешний ключ.

Требования к значениям столбцов внешнего ключа отличаются в зависимости от того, ссылается ли внешний ключ дочерней таблицы на первичный или на уникальный ключ родительской таблицы.

Если внешний ключ ссылается на первичный ключ, то либо набор значений столбцов внешнего ключа должен в точности соответствовать набору значений первичного ключа какой-нибудь из строк родительской таблицы, либо все значения столбцов, входящих в состав внешнего ключа дочерней таблицы, должны иметь пустое значение.

Если же внешний ключ ссылается на уникальный ключ родительской таблицы, то допустима также ситуация, когда только некоторые столбцы внешнего ключа имеют пустое значение. При этом, разумеется, в родительской таблице должна быть строка, в которой уникальный ключ имеет такой же набор пустых и непустых значений ключевых столбцов, что и во внешнем ключе.

Необязательные предложения ON DELETE и ON UPDATE определяют, что будет происходить с дочерней таблицей, соответственно, при удалении строки родительской таблицы или при изменении ключевых данных в родительской таблице. Описание соответствующих действий см. ранее в этой главе, где обсуждалось ограничение внешнего ключа для одного столбца.

Во многих случаях ограничения внешних ключей для таблиц базы данных удобнее задавать не при создании таблицы, а при выполнении изменения таблиц после того, как все таблицы уже созданы (при помощи оператора ALTER TABLE). Это избавит вас от необходимости жестко определять порядок создания таблиц в базе данных. В некоторых случаях, когда между двумя таблицами существуют перекрестные ссылки, единственным способом описать внешние ключи будет именно последующее выполнение операторов изменения таблиц. Подробнее см. в разделе 5.2 Изменение таблиц.

### Ограничение CHECK

Это ограничение задает условия, которым должны удовлетворять значения столбцов данной таблицы при помещении в таблицу новой строки (оператор INSERT) или при изменении (оператор UPDATE) отдельных столбцов существующей строки таблицы. Синтаксис ограничения:

```
CHECK (<условие таблицы>)
```

Варианты и правила использования условий таблицы полностью совпадают с условиями CHECK столбца, описанными ранее в этой главе.

## 5.2 Изменение таблиц

Описание существующих в базе данных таблиц (характеристик столбцов, их порядка, наличие различных ключей или ограничения CHECK) можно изменять после создания таблиц.

Изменение структуры таблиц, уже заполненных данными, является одним из наиболее опасных действий, которое часто приводит к исключениям базы данных или к потере существующих в

таблице данных.

Для изменения структуры существующих таблиц используется оператор ALTER TABLE. Изменять таблицу может ее владелец, администратор и пользователь с ролью ALTER ANY TABLE.

Синтаксис оператора ALTER TABLE представлен в листинге 5.10.

**Листинг 5.10. Синтаксис оператора изменения таблицы ALTER TABLE**

```
ALTER TABLE <имя таблицы> <операция изменения> [, <операция изменения>...];
```

В одном операторе можно выполнить произвольное количество изменений в таблице. Различные операции по изменению существующей таблицы отделяются друг от друга запятыми.

Синтаксис такой операции изменения существующей таблицы показан в листинге 5.11.

**Листинг 5.11. Синтаксис операции изменения в операторе ALTER TABLE**

```
<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 }
```

Некоторые изменения структуры таблицы увеличивают счётчик форматов, закреплённый за каждой таблицей. Количество форматов для каждой таблицы ограничено значением 255. После того, как счётчик форматов достигнет этого значения, вы не сможете больше менять структуру таблицы.

Для сброса счётчика форматов необходимо сделать резервное копирование и восстановление базы данных.

## Добавление нового столбца

Для добавления нового столбца в существующую в базе данных таблицу следует ввести в операторе изменения таблицы ALTER TABLE следующую конструкцию:

```
ADD <определение столбца>
```

Синтаксис определения столбца представлен в листинге 5.12.

**Листинг 5.12. Синтаксис определения столбца**

```
<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 BY DEFAULT AS IDENTITY [(START WITH <стартовое значение>)]
   [<ограничение столбца>]
```

В определении добавляемого столбца присутствует как собственно описание характеристик столбца, так и возможное ограничение столбца. В операторе можно задать значение по умолчанию для столбца (предложение DEFAULT), задать ограничение недопустимости пустого значения NOT NULL, добавить различные ограничения столбца или установить порядок сортировки (предложение COLLATE). Синтаксис и семантика того, как в таблице описываются столбцы и их ограничения см. в подразделе 5.1 Определение столбца этой главы.

Если в таблице, куда добавляется новый столбец, уже в базе данных существуют строки данных, то к каждой строке присоединяется новый столбец с пустым значением NULL. На это значение NULL не влияет и наличие в описании вновь добавляемого столбца значения по умолчанию DEFAULT. Не оказывает также никакого влияния и присутствие характеристики в добавляемом столбце NOT NULL. Новый столбец становится последним столбцом в структуре таблицы.

## Добавление ограничения таблицы

Для добавления нового ограничения таблицы нужно в операторе изменения таблицы ALTER TABLE ввести следующий оператор:

```
ADD <ограничение таблицы>
```

Ограничение таблицы описывается следующим синтаксисом (листинг 5.13):

**Листинг 5.13. Синтаксис ограничения таблицы**

```
<ограничение таблицы> ::= [CONSTRAINT <имя ограничения>] {
    PRIMARY KEY (<имя столбца> [, <имя столбца> ...]) [<предложение USING>]
  | UNIQUE (<имя столбца> [, <имя столбца> ...]) [<предложение USING>]
  | FOREIGN KEY (<имя столбца> [, <имя столбца> ...])
    REFERENCES <имя таблицы> (<имя столбца>[,<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>) }
```

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.

Все синтаксические конструкции и подробное описание ограничений таблицы представлены ранее в подразделе 5.1 Определение ограничений таблицы этой главы.

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.** 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.** Добавление проверочного ограничения и внешнего ключа:

```
ALTER TABLE JOB
ADD CONSTRAINT CHK_SALARY CHECK (MIN_SALARY < MAX_SALARY),
ADD FOREIGN KEY (JOB_COUNTRY)
REFERENCES COUNTRY (COUNTRY);
```

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.

## 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 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.

## 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.

Следует проявлять осторожность только при удалении ограничений первичного и уникального ключа, поскольку на такие ограничения могут быть ссылки внешнего ключа в других дочерних таблицах. Другие ограничения — ограничения внешнего ключа и ограничения CHECK не имеют никаких зависимостей, делающих невозможным их удаление.

## Изменение существующего столбца

При использовании оператора ALTER TABLE есть несколько вариантов изменения характеристик существующего столбца таблицы с помощью предложения ALTER [COLUMN]:

- изменение имени;
- изменение типа данных;
- изменение позиции столбца в списке столбцов таблицы;
- удаление значения по умолчанию столбца;
- добавление значения по умолчанию столбца;
- изменение типа и выражения для вычисляемого столбца;
- изменение столбцов идентификации.

Само предложение ALTER [COLUMN] выглядит следующим образом:

**Листинг 5.14. Синтаксис операции изменения в операторе ALTER [COLUMN]**

```
ALTER [COLUMN] <имя столбца> <модификация столбца>

<модификация столбца> ::= <мод-я обычного столбца> | <мод-я вычисляемого столбца> |
<мод-я идентификационного столбца>

<модификация обычного столбца> ::=
    TO <новое имя столбца>
  | POSITION <новая позиция>
  | TYPE { <тип данных> | <имя домена> }
  | SET DEFAULT { <литерал> | NULL | <контекстная переменная>}
  | DROP DEFAULT
  | SET NOT NULL
  | DROP NOT NULL

<модификация вычисляемого столбца> ::=
    TO <новое имя столбца>
  | POSITION <новая позиция>
  | [TYPE <тип данных>] {GENERATED ALWAYS AS | COMPUTED [BY]} (<выражение>)

<модификация идентификационного столбца> ::=
    TO <новое имя столбца>
  | POSITION <новая позиция>
  | RESTART [ WITH <стартовое значение> ]
```

## Изменение имени

Для изменения имени столбца в операторе изменения таблицы `ALTER TABLE` используется конструкция:

```
ALTER [COLUMN] <имя столбца> TO <новое имя столбца>
```

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, созданных 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.

## Изменение типа данных

Для изменения типа данных столбца существующей в базе данных таблицы используется следующая синтаксическая конструкция в операторе изменения таблицы `ALTER TABLE`:

```
ALTER [COLUMN] <имя столбца> TYPE <новый тип данных>
```

Если столбец был объявлен как массив, то изменить ни его тип, ни размерность нельзя.

Тип `BLOB` у столбца можно изменить только на `BLOB` такого же подтипа.

Нельзя изменить тип данных у столбца, который принимает участие в связке внешний ключ / первичный (уникальный) ключ. В остальных случаях изменение типа данных возможно, однако следует помнить, что это может привести к большим сложностям при дальнейшей эксплуатации базы данных, если перед таким изменением таблица была заполнена данными.

При таком изменении таблица будет содержать все существовавшие на момент изменения типа данных строки в одной структуре, а вновь добавляемые строки будут иметь иную структуру. Попытки изменения ранее существующих строк в контексте текущей транзакции чаще всего будут приводить к исключениям базы данных. Даже попытки выборки данных из этой таблицы, скорее всего, приведут к исключениям.

Если действительно необходимо изменить тип данных отдельного столбца существующей таблицы, то следует создать новую временную таблицу, куда нужно скопировать все уже введенные данные изменяемой таблицы. В старой таблице нужно удалить все записи. После этого можно изменять тип данных столбца, восстанавливать данные из временной таблицы со всеми необходимыми преобразованиями данных измененного столбца и удалять временную таблицу.

## 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 у вас 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.

Номер позиции — это положительное число. Столбцы в таблицах нумеруются, начиная с позиции один. Если указать номер позиции, превышающий количество столбцов в таблице, то никакие

изменения в таблице выполнены не будут, исключений базы данных не возникнет. При задании же числа меньше единицы будет выдано сообщение об ошибке, такое изменение не будет выполнено.

### 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 — доменное значение перекроет это удаление.

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

### 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 для столбца всегда перекрывает доменное значение по умолчанию.

### Добавление и удаление ограничения NOT NULL

Предложение `SET NOT NULL` добавляет ограничение `NOT NULL` для столбца таблицы.

```
ALTER [COLUMN] <имя столбца> SET NOT NULL
```

Успешное добавление ограничения `NOT NULL` происходит, только после полной проверки данных таблицы, для того чтобы убедится что столбец не содержит значений `NULL`.

Явное ограничение `NOT NULL` на столбце, базирующегося на домене, преобладает над установками домена. В этом случае изменение домена для допустимости значения `NULL`, не распространяется на столбец таблицы.

Предложение `DROP NOT NULL` удаляет ограничение `NOT NULL` для столбца таблицы. Если столбец основан на домене с ограничением `NOT NULL`, то ограничение домена перекроет это удаление.

### Изменение типа и выражения для вычисляемого столбца

Для вычисляемых столбцов допустимо изменить тип и выражение вычисляемого столбца:

```
ALTER [COLUMN] <имя столбца> [TYPE <тип данных>] {GENERATED ALWAYS AS | COMPUTED
[BY]} (<выражение>)
```

Невозможно изменить обычный столбец на вычисляемый и наоборот.

### Изменение столбцов идентификации

Для столбцов идентификации позволено изменять начальное значение. Если указано только предложение `RESTART`, то происходит сброс значения генератора в ноль. Необязательное предложение `WITH` позволяет указать для нового значения внутреннего генератора отличное от нуля значение. Невозможно изменить обычный столбец на столбец идентификации и наоборот.

### 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 таблица вызывается с правами текущего пользователя. Задание ключевого слова `DEFINER` означает, что таблица вызывается 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`.

Предложение `DROP SQL SECURITY` удаляет эту опцию, указанную при создании.

## 5.3 Удаление таблиц

Для удаления существующей таблицы используется оператор `DROP TABLE`. Удалять таблицу может ее владелец, администратор и пользователь с привилегией `DROP ANY TABLE`.

Синтаксис оператора представлен в листинге 5.15.

**Листинг 5.15. Синтаксис оператора удаления таблицы `DROP TABLE`**

```
DROP TABLE <имя таблицы>;
```

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.

## 5.4 Пересоздание таблицы

Таблица пересоздается оператором `RECREATE TABLE`. Синтаксис оператора представлен в листинге 5.16.

**Листинг 5.16. Синтаксис оператора создания таблицы `RECREATE TABLE`**

```
RECREATE TABLE <имя таблицы>
  [EXTERNAL [FILE] '<спецификация файла>']
  (<определение столбца> [, {<определение столбца> | <ограничение таблицы>}...])
  [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`.

## 5.5 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, поми-

мо 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 5.17):

**Листинг 5.17. 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 6

# Работа с генераторами

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.

## 6.1 Создание генератора

Для создания генератора используется оператор SQL `CREATE GENERATOR/SEQUENCE`. Синтаксис оператора показан в листинге 6.1.

**Листинг 6.1. 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.

Имя генератора должно быть уникальным среди имен всех генераторов базы данных и должно содержать до 31 символа.

В момент создания последовательности ей устанавливается значение, указанное в необязательном предложении `START WITH`. Если предложение `START WITH` отсутствует, то последовательности устанавливается значение равное 0.

Необязательное предложение `INCREMENT [BY]` позволяет задать шаг приращения для оператора `NEXT VALUE 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 предложении `INCREMENT BY`.

Создавать последовательности могут администраторы и те пользователи, у кого есть привилегия `CREATE SEQUENCE (CREATE GENERATOR)`. Пользователь, создавший последовательность, становится её владельцем.

## 6.2 Изменение значения генератора

Можно явно в любой момент времени установить новое значение генератора, выполнив оператор `SET GENERATOR` (листинг 6.2).

**Листинг 6.2. 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 6.3). Именно этот вариант рекомендуется использовать в настоящей версии Ред База Данных.

**Листинг 6.3. Альтернативный синтаксис оператора изменения значения генератора ALTER SEQUENCE**

```
ALTER SEQUENCE <имя генератора> RESTART WITH <значение>
  [RESTART [WITH <значение>]]
  [INCREMENT [BY] <приращение>];
```

Предложение RESTART WITH позволяет установить значение последовательности. Предложение RESTART может быть использовано самостоятельно (без WITH) для перезапуска значения последовательности с того значения, с которого был начат старт генерации значений или предыдущий рестарт.

Предложение INCREMENT [BY] позволяет изменить шаг приращения последовательности для оператора NEXT VALUE 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 (ALTER GENERATOR) и SET GENERATOR могут выполнять владельцы последовательностей, администраторы и пользователи с привилегией ALTER ANY SEQUENCE (ALTER ANY GENERATOR).

## 6.3 Создание нового или изменение существующего генератора

С помощью оператора CREATE OR ALTER GENERATOR (SEQUENCE) можно создать новую или изменить существующую последовательность:

**Листинг 6.4. 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.

## 6.4 Удаление генератора

Генератор можно удалить, используя оператор SQL `DROP GENERATOR / SEQUENCE`. Его синтаксис представлен в листинге 6.5.

**Листинг 6.5. Синтаксис оператора удаления генератора `DROP GENERATOR /SEQUENCE`**

```
DROP {GENERATOR | SEQUENCE) <имя генератора>;
```

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` (`DROP ANY GENERATOR`).

Практическое использование генераторов для формирования значений первичного ключа в таблицах см. в главе 8 «Операторы DML».

## 6.5 Пересоздание генератора

Последовательность можно пересоздать с помощью оператора `RECREATE GENERATOR (SEQUENCE)`:

**Листинг 6.6. 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.

## 6.6 Примечание к генератору

Для генератора также можно создать примечание, используя следующий синтаксис оператора `COMMENT` (см. листинг 6.7).

**Листинг 6.7. 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 7

# Работа с индексами

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.

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 индексов 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.

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.

## 7.1 Создание индекса

Для создания индекса для существующей таблицы базы данных используется оператор `CREATE INDEX`. Его синтаксис представлен в листинге 7.1.

**Листинг 7.1. Синтаксис оператора создания индекса `CREATE INDEX`**

```
CREATE [UNIQUE] [ASC[ENDING] | DESC[ENDING]]
INDEX <имя индекса> ON <таблица>
{(<столбец> [, <столбец> ...]) | COMPUTED BY (<выражение>)};
```

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 5 «Работа с таблицами») 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.

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.

**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);
```

## 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** | | | | |
|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 6 |
| **4096** | 1015 | 507 | 338 | 253 | 169 |
| **8192** | 2039 | 1019 | 679 | 509 | 339 |
| **16384** | 4087 | 2043 | 1362 | 1021 | 682 |

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 |

Создать индекс может только владелец таблицы, для которой создан индекс, администратор и пользователь с привилегией ALTER ANY TABLE.

## 7.2 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 7.2. 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 7.2. Синтаксис оператора изменения состояния индекса ALTER INDEX**

```
ALTER INDEX <имя индекса> {ACTIVE | INACTIVE};
```

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 в активное состояние система заново полностью создает весь индекс.

Состояние индекса может изменять только владелец таблицы, для которой создан индекс, администратор и любой пользователь с привилегией ALTER ANY TABLE. Подробнее см. в документе «Руководство администратора».

Состояние системного индекса может изменять только пользователь SYSDBA и владелец базы данных.

При изменении состояния системного индекса может возникнуть ошибка "Cannot find system index, try to reconnect", если в этот момент он используется.

**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;
```

## 7.3 Удаление индекса

Для удаления индекса, созданного пользователем, используется оператор DROP INDEX. Его синтаксис представлен в листинге 7.3.

**Листинг 7.3. 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;
```

## 7.4 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.

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 7.4.

**Листинг 7.4. 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.

## 7.5 Примечание индекса

Для индекса можно создать примечание, используя следующий вариант оператора COMMENT (листинг 7.5).

**Листинг 7.5. 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 8

# 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 системе управления базами данных Ред База Данных. Он позволяет выбирать данные из одной или более 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 более 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 10 «Транзакции».

## 8.1 SELECT

Синтаксис оператора достаточно сложный. Он представлен в листинге 8.1. Далее в подразделе дается краткое описание структуры оператора `SELECT`, о полном описании всех предложении рассказывается в следующих подразделах этой главы.

**Листинг 8.1. Синтаксис оператора выборки данных `SELECT`**

```
[WITH [RECURSIVE] <CTE> [, <CTE> ...]]
SELECT
  [FIRST <значение>] [SKIP <значение>]
  [DISTINCT | ALL]
  <выходное поле> [, <выходное поле>]
FROM
  <источники>
  [<соединяемые источники>]
[WHERE <условие выборки>]
[GROUP BY <условие группирование выбранных данных>
 [HAVING <условие выборки>]]
```

```
[UNION [DISTINCT | ALL] <другой набор данных>]
[PLAN <выражение для плана поиска>]
[ORDER BY <выражение для порядка выборки>]
[OPTIMIZE FOR {FIRST | ALL} ROWS]
[   ROWS <m> [TO <n>]
  | [OFFSET <n> {ROW | ROWS}] [FETCH {FIRST | NEXT} [<m>] {ROW | ROWS} ONLY] ]
[FOR UPDATE [OF <имя столбца> [, <имя столбца>]...]
[WITH LOCK]
[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).

Сразу после ключевого слова SELECT могут следовать предложения FIRST и SKIP, позволяющие определить количество помещаемых в результирующий набор данных строк, выбранных на основании условий выборки (предложения ON, UNION и WHERE).

Ключевое слово DISTINCT указывает, что в выходной набор данных не помещаются 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 «Предложение WHERE» 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 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 10 «Транзакции».

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).

## Предложение 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 8.2:

**Листинг 8.2. Синтаксис предложения WITH RECURSIVE**

```
WITH [RECURSIVE] <CTE> [, <CTE> ...]

<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 представлен в листинге 8.3.

**Листинг 8.3. Пример не рекурсивного WITH RECURSIVE**

```
WITH DEPT_YEAR_BUDGET AS (
   SELECT
     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` представлен в листинге 8.4.

**Листинг 8.4. Пример рекурсивного `WITH RECURSIVE`**

```
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
   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.

## Список выбора

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 и каков список выбора — какие столбцы исходных таблиц должны присутствовать в выходном наборе данных.

**Листинг 8.5. Синтаксис списка выбора в операторе SELECT**

```
SELECT
  [FIRST <значение>] [SKIP <значение>]
  [DISTINCT | ALL]
  <выходное поле> [, <выходное поле>]
FROM ...

<выходное поле> ::= {
   [<спецификатор>.]*
  | <список выбора> [COLLATE <порядок сортировки>] [[AS] <псевдоним>] }

<список выбора> ::= {
    [<спецификатор>.]<имя столбца таблицы или представления>[[<элемент массива>]]
  | [<спецификатор>.]<выходной параметр селективной хранимой процедуры>
  | <константа>
  | NULL
  | <контекстная переменная>
  | <обычная внутренняя или агрегатная функция>
  | <функция UDF>
  | <подзапрос, возвращающий единственное скалярное значение>
  | <конструкция CASE>
  | NEXT VALUE FOR <имя генератора>
  | <любое выражение, возвращающее единственное значение> }

<спецификатор> ::= имя таблицы (представления) или псевдоним таблицы
(представления, хранимой процедуры, производной таблицы)
```

Необязательный вариант FIRST задает, что в выходной набор данных должно быть помещено указанное количество первых выбранных строк. Вариант может использоваться самостоятельно или вместе с вариантом SKIP. Подробнее об этих ключевых словах рассказано ниже в этом подразделе.

Далее может следовать одно из необязательных ключевых слов DISTINCT или ALL. Ключевое слово DISTINCT определяет, что все выбранные данные, указанные в списке выбора, должны отличаться друг от друга, дубликаты строк не будут помещаться в результирующий набор данных. Ключевое слово ALL, принимаемое 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.

Сразу после ключевых слов DISTINCT или ALL идет сам список выбора, определяющий, какие столбцы из каких таблиц должны помещаться в результирующий набор данных.

Символ «*» означает, что в результирующем наборе данных должны присутствовать все столбцы исходной таблицы (исходных таблиц). Такой вариант следует применять только в случае простого выбора данных из одной таблицы. При выборе данных из нескольких таблиц (при соединении нескольких таблиц) или в случае, когда одна исходная таблица имеет очень много столбцов, следует явно задавать имена нужных для дальнейшей обработки столбцов.

Если выбираемый столбец является массивом, то нужно явно указать, какой именно элемент массива выбирается. Не рекомендуется использовать массивы в реляционных базах данных. Если же их использование диктуется эффективностью обработки данных, то следует проявлять осторожность при их применении.

Значение в списке выбора может быть именем столбца (используется чаще всего), константой соответствующего вида, любым довольно сложным выражением, обращением к внутренней функции (COUNT, AVG, MIN, MAX, TRIM, SUBSTRING, SUM, CAST, UPPER, LOWER, GEN_ID, EXTRACT и др.) Это может быть обращение к конструкции NEXT VALUE FOR или к функции, определенной пользователем, UDF (User Defined Function — см. приложение Г), пустым значением NULL или именем пользователя, соединенного в настоящий момент с базой данных (USER). Подробнее об использовании внутренних функций см. в разделе «Предложение WHERE» этой главы. Элементом списка может быть и одиночный оператор SELECT, обращающийся к любой таблице базы данных и возвращающий в точности одно значение одного столбца или пустое значение NULL. Такой оператор обязательно должен быть заключен в круглые скобки.

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:

```
COUNTRY.CODCOUNTRY
```

> Для столбцов таблицы самого высокого уровня в операторе SELECT имя (псевдоним) таблицы можно не указывать.

Если для таблицы был указан псевдоним (см. далее), то везде следует использовать только псевдоним, но не имя таблицы.

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,

```
COUNTRY.CODCOUNTRY AS "Код страны"
```

Если для отдельных столбцов были заданы псевдонимы, то при использовании для отображения строк таблицы утилиты isql или любой программы графического интерфейса 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.

## Использование ключевых слов FIRST и SKIP

Выбираемые строки оператором SELECT определяются условием в предложении WHERE и в предложениях ON, если в операторе используется соединение таблиц. Однако в выходной набор данных могут попасть не все эти строки, а меньшее их количество. Для уменьшения объема выборки используются либо ключевые слова FIRST и/или SKIP, либо предложение ORDER BY вместе с предложением ROWS, OFFSET, FETCH. Для этих целей может использоваться только одна из перечисленных конструкций.

Необязательное ключевое слово FIRST задает количество первых записей полученного в результате выборки набора данных, которые попадут в результирующий набор данных. Строки нумеруются, начиная с единицы. Если полученный набор данных содержит меньше чем указано в FIRST строк, то в результирующий набор данных будут помещены только существующие строки без выдачи каких-либо сообщений.

Необязательное ключевое слово SKIP указывает, что заданное количество первых строк полученного набора данных не попадет в результирующий набор данных. Если набор данных содержит меньше, чем указано в SKIP строк, то результирующий набор данных будет пустым. Никаких сообщений выдаваться не будет.

В одном операторе SELECT можно указать сразу и FIRST, и SKIP или одно из этих ключевых слов.

Значениями здесь могут быть как числа, так и любые сколь угодно сложные выражения, возвращающие число. Если возвращается дробное число, то десятичные знаки просто отбрасываются без округления. Если в качестве значений используются выражения, то все выражение должно заключаться в круглые скобки. Если в выражении используется оператор SELECT, то он дополнительно должен быть заключен в круглые скобки.

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 FIRM) / 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 существует предложение ROWS.

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 помещена только одна десятая строка.

Количество помещаемых в результирующий набор данных строк может также задаваться (корректироваться) при использовании предложений ORDER BY и ROWS, а также предложений OFFSET, FETCH.

> FIRST, SKIP и ROWS используются только в Ред базе данных, они не включены в стандарт SQL. Рекомендуется использовать OFFSET, FETCH.

## Предложение FROM

Предложение FROM задает таблицу или список таблиц, из которых осуществляется выборка данных. Здесь также может быть указано имя представления, имя хранимой процедуры выбора, которая получает данные из одной или более таблиц, или общее табличное выражение (CTE). Эти таблицы, представления, процедуры могут комбинироваться с использованием разнообразных видов соединений (JOIN).

**Листинг 8.6. Синтаксис предложения FROM в операторе SELECT**

```
SELECT ...
FROM
   <источники>
  [<соединяемые источники>]

<источники> ::= {
    <таблица>
  | <представление>
  | <селективная хранимая процедура> [(<аргументы>)]
  | <производная таблица>
  | <псевдоним CTE>
} [[AS] <псевдоним>]

<производная таблица> ::= (<SELECT запрос>) [[AS] <псевдоним производной таблицы>]
[(<псевдоним столбца производной таблицы> [, <псевдоним столбца>])]

<соединяемые источники> ::= <соединение> [ <соединение> ...]

<соединение> ::=
  [ <вид соединения>] JOIN <источники> <условие соединения>
  | NATURAL [<вид соединения>] JOIN <источники>
  | {CROSS JOIN | ,} <источники>

<вид соединения> ::= INNER | {LEFT | RIGHT | FULL} [OUTER]

<условие соединения> ::= ON <условие> | USING (<список столбцов>)
```

В предложении FROM существует возможность указания имен нескольких таблиц (представлений, хранимых процедур выбора), разделенных запятыми, где каждая из последующих 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 такую возможность неявного соединения таблиц. Следует явно задавать соединяемые таблицы (в том числе, полученные из представлений или из хранимых процедур выбора) с использованием ключевого слова JOIN и условие соединения, указанное в предложении ON. Относительно соединения таблиц см. раздел «Соединение таблиц (JOIN)» в этой главе.

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;
```

Здесь для таблицы задан псевдоним, по этой причине нельзя в конструкциях оператора SELECT использовать имя таблицы. Тут нужно или в списке выбора использовать не имя, а псевдоним таблицы, или вообще не задавать для таблицы псевдоним. Правильно будут выполнены операторы

```
SELECT C.CODCOUNTRY
FROM COUNTRY C;
```

или

```
SELECT COUNTRY.CODCOUNTRY
FROM COUNTRY;
```

## Выборка из таблицы или представления

При выборке из таблицы или представления предложение FROM не требует ничего кроме его имени. Псевдоним (алиас) может быть полезен или даже необходим при использовании подзапросов, которые соотнесены с главным запросом (обычно подзапросы являются коррелированными).

```
SELECT id, name, sex, age
FROM actors
WHERE state = 'Ohio';
```

## Выборка из селективной хранимой процедуры

Селективная хранимая процедура (т.е. с возможностью выборки) должна удовлетворять следующим условиям:

- Содержать, по крайней мере, один выходной параметр;
- Использовать ключевое слово SUSPEND таким образом, чтобы вызывающий запрос мог выбирать выходные строки одну за другой, также как выбираются строки 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 опциональных 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 База 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 8.7.

**Листинг 8.7. Синтаксис производной таблицы**

```
<производная таблица> ::= (<оператор SELECT>)
    [[AS] <псевдоним таблицы>]
    [(<псевдоним столбца> [, <псевдоним столбца>...])]
```

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 более 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 именно имен псевдонимов, если они были указаны.

Следующий оператор выбирает список имен и внутренних идентификаторов всех несистемных таблиц из системной таблицы RDB$RELATIONS. Пример использования производной таблицы:

**Пример 8.1**

```
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 обычные таблицы. CTE доступны только в рамках основного запроса.

Подробно CTE описываются в подразделе «Предложение WITH RECURSIVE».

Приведем пример использования. 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.

**Пример 8.2**

```
WITH vars (b, D, denom) AS (
        SELECT b, b*b - 4*a*c, 2*a
        FROM coeffs ),
     vars2 (b, D, denom, sqrtD) AS (
        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
```

## Соединение таблиц (JOIN)

Средства соединения таблиц (JOIN) позволяют выбрать данные из нескольких таблиц. Это более мощные средства, чем объединение (UNION) таблиц. При проектировании базы данных в процессе нормализации таблиц часто одна таблица разделяется на несколько таблиц, имеющих более простую структуру. Обычно средства соединения используют отношение, установленное между родительской и дочерней таблицей при помощи связки внешний ключ/первичный (уникальный) ключ. Эти же возможности можно использовать и для любых других существующих в реальной жизни связей между таблицами.

Соединения объединяют данные из двух источников в один набор данных. Соединение данных осуществляется для каждой строки и обычно включает в себя проверку условия соединения для того, чтобы определить, какие строки должны быть объединены и оказаться в результирующем наборе данных. Результат соединения также может быть соединён с другим набором данных с помощью следующего соединения.

Существует внешнее (OUTER) и внутреннее (INNER) соединения, а также перекрестное соединение, или декартово произведение строк таблиц (CROSS). Внешнее соединение может быть левым (LEFT), правым (RIGHT) и полным (FULL).

В предложении FROM должна присутствовать, как минимум, одна основная таблица, с которой могут соединяться другие таблицы. Синтаксис соединяемой таблицы следующий:

```
<соединение> ::=
  [ <вид соединения>] JOIN <источники> <условие соединения>
  | NATURAL [<вид соединения>] JOIN <источники>
  | {CROSS JOIN | ,} <источники>
<вид соединения> ::= INNER | {LEFT | RIGHT | FULL} [OUTER]
<условие соединения> ::= ON <условие> | 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.

### Внутреннее соединение

Для внутреннего (INNER) соединения вид соединения можно не указывать — соединение является внутренним по умолчанию. Иногда в литературе такое соединение называют естественным соединением (natural join).

Результатом внутреннего соединения двух таблиц является так называемое декартово произведение двух множеств (строк двух таблиц) В этом случае к каждой строке первой таблицы присоединяются все строки второй таблицы. В выходной набор данных попадают только те строки полученного декартового произведения, которые соответствуют условию соединения, заданному в предложении ON.

**Пример.** Существует таблица PEOPLE, содержащая данные о людях, и таблица STAFF, где описаны сотрудники различных организаций.

Таблица для хранения сведений о людях создавалась оператором CREATE TABLE:

**Пример 8.3**

```
CREATE TABLE PEOPLE (
   COD INTEGER NOT NULL, /* Код человека */
   NAME1 CHAR(15), /* Имя */
   NAME2 CHAR(15), /* Отчество */
   NAME3 CHAR(20), /* Фамилия */
   BIRTHDAY DATE, /* Дата рождения */
   SEX CHAR(1) DEFAULT '0', /* Пол: 0 - мужской, 1 - женский.*/
   FULLNAME COMPUTED BY (NAME3 || ' '|| NAME1 || ' ' || NAME2),
   CODMOTHER INTEGER, /* Ссылка на мать */
   CODFATHER INTEGER, /* Ссылка на отца */
   CODOTHERHALF INTEGER, /* Ссылка на супруга */
   CONSTRAINT PK_PEOPLE PRIMARY KEY (COD),
   CONSTRAINT CH_PEOPLE CHECK (SEX IN ('0', '1')),
   CONSTRAINT FK1_PEOPLE
     FOREIGN KEY (CODMOTHER) REFERENCES PEOPLE (COD)
     ON DELETE SET NULL,
   CONSTRAINT FK2_PEOPLE
     FOREIGN KEY (CODFATHER) REFERENCES PEOPLE (COD)
     ON DELETE SET NULL,
   CONSTRAINT FK3_PEOPLE
     FOREIGN KEY (CODOTHERHALF) REFERENCES PEOPLE (COD)
     ON DELETE SET NULL
);
CREATE SEQUENCE GEN_PEOPLE;
```

Последней строкой здесь создается генератор, который используется для получения значений искусственного первичного ключа при помещении новой строки в базу данных.

Для создания таблицы, хранящей сведения о сотрудниках различных организаций, используется следующий скрипт:

**Пример 8.4**

```
CREATE TABLE STAFF (
   COD INTEGER NOT NULL, /* Код сотрудника */
   CODPEOPLE INTEGER, /* Код человека */
   CODORG INTEGER, /* Код организации */
   DUTIES CHAR(40), /* Должность */
   SALARY DECIMAL(8,2), /* Оклад */
   NET_SALARY COMPUTED BY (SALARY * .87),
   CONSTRAINT PK_STAFF PRIMARY KEY (COD),
   CONSTRAINT FK1_STAFF
     FOREIGN KEY (CODPEOPLE) REFERENCES PEOPLE (COD)
     ON DELETE CASCADE,
   CONSTRAINT FK2_STAFF
     FOREIGN KEY (CODORG) REFERENCES ORGANIZATION (COD)
     ON DELETE CASCADE
);
CREATE SEQUENCE GEN_STAFF;
```

В списке сотрудников отсутствуют сведения о фамилиях людей (это соответствует требованиям второй нормальной формы реляционных баз данных). Связь с таблицей людей осуществляется при помощи внешнего ключа FK1_STAFF, ссылающегося на таблицу PEOPLE.

Чтобы выбрать список всех сотрудников, добавив при этом фамилию, имя и отчество из таблицы людей (это вычисляемый столбец FULLNAME в таблице PEOPLE) можно использовать следующий оператор SELECT, задав внутреннее соединение:

**Пример 8.5**

```
SELECT
   P.FULLNAME AS "Сотрудник" ,
   DUTIES AS "Должность"
FROM STAFF S
   INNER JOIN PEOPLE P
     ON S.CODPEOPLE = P.COD
ORDER BY "Сотрудник";
```

Здесь формируется нужный набор данных, содержащий необходимые сведения о сотрудниках. Условием соединения является равенство внешнего ключа таблицы персонала первичному ключу таблицы людей, где содержатся необходимые дополнительные сведения о людях.

Обратите внимание на то, что в предложении ORDER BY использован псевдоним столбца.

Точно такой же результат можно получить, выполнив следующий оператор SELECT:

**Пример 8.6**

```
SELECT
   P.FULLNAME AS "Сотрудник" ,
   DUTIES AS "Должность"
FROM STAFF S, PEOPLE P
WHERE S.CODPEOPLE = P.COD
ORDER BY "Сотрудник";
```

Это неявное внутреннее соединение. Условие соединения в этом случае задается в предложении WHERE.

Здесь выполнялось внутреннее соединение двух таблиц. Однако таблица сотрудников содержит ссылку и на организацию, где работает человек. Список организаций содержится в таблице FIRM, которая создается при помощи следующего скрипта:

**Пример 8.7**

```
CREATE TABLE FIRM (
   COD INTEGER NOT NULL, /* Код организации */
   CODCTR CHAR(3), /* Код страны */
   CODREG CHAR(2), /* Код региона */
   CODAREA CHAR(2), /* Код района */
   LOCATION CHAR(1) DEFAULT '0',/* Признак адреса: 0 - областной центр,
                                   1 - районный центр, 2 - район.*/
   NAME CHAR(60), /* Название организации */
   CONSTRAINT PK_FIRM PRIMARY KEY (COD),
   CONSTRAINT CH_FIRM CHECK (LOCATION IN ('0', '1', '2')),
   CONSTRAINT FK1_FIRM
     FOREIGN KEY (CODCTR, CODREG) REFERENCES REFREG (CODCTR, CODREG)
     ON DELETE SET NULL
     ON UPDATE CASCADE,
   CONSTRAINT FK2_FIRM
```

```
    FOREIGN KEY (CODFORMORG) REFERENCES REFFORMORG (COD)
    ON DELETE SET NULL
    ON UPDATE CASCADE
);
CREATE SEQUENCE GEN_FIRM;
```

Для получения сведений не только о фамилии, имени и отчестве человека, но и о названии организации, в которой он работает, следует использовать внутреннее соединение уже трех таблиц — STAFF, PEOPLE и FIRM:

**Пример 8.8**

```
SELECT
   P.FULLNAME AS "Сотрудник" ,
   F.NAME AS "Организация" ,
   DUTIES AS "Должность"
FROM STAFF S
   INNER JOIN PEOPLE P
     ON S.CODPEOPLE = P.COD
   INNER JOIN FIRM F
     ON S.CODORG = F.COD
ORDER BY "Сотрудник";
```

Такой же результат можно получить, выполнив следующий оператор SELECT, где таблицы, участвующие во внутреннем соединении, просто перечислены в предложении FROM, а условие соединения присутствует в предложении WHERE:

**Пример 8.9**

```
SELECT
   P.FULLNAME AS "Сотрудник" ,
   DUTIES AS "Должность"
   F.NAME AS "Организация" ,
FROM STAFF S, PEOPLE P, FIRM F
WHERE (S.CODPEOPLE = P.COD) AND (S.CODORG = F.COD)
ORDER BY "Сотрудник";
```

Можно выполнять внутреннее соединение таблицы с той же самой таблицей. В следующем примере показано двойное внутреннее соединение одной и той же таблицы с самой собой. К таблице людей добавляются фамилии матери и отца, получаемые из той же самой таблицы. Посмотрите на один из предыдущих примеров, где описана таблица людей PEOPLE (Пример 8.3). В этой таблице присутствуют столбцы, являющиеся внешними ключами, ссылающимися на ту же самую таблицу PEOPLE. Эти внешние ключи являются ссылками на запись отца (CODFATHER) и на мать (CODMOTHER) конкретного человека. Получить необходимую таблицу людей со сведениями о родителях можно, выполнив следующий оператор:

**Пример 8.10**

```
SELECT
   PG.FULLNAME AS "Фамилия, имя, отчество" ,
   PM.NAME3 AS "Мать
   PF.NAME3 AS "Отец"
FROM PEOPLE PG /* Главная таблица */
   INNER JOIN PEOPLE PM /* Мать */
     ON PG.CODMOTHER = PM.COD
```

```
    INNER JOIN PEOPLE PF /* Отец */
      ON PG.CODFATHER = PF.COD
ORDER BY PG.FULLNAME;
```

В результирующем наборе данных будут присутствовать только те строки, для которых найдены все соответствия. Строки, содержащие в столбцах ссылок на отца и/или на мать пустые значения NULL, в результирующий набор данных не попадут. Не всегда такое поведение системы будет отвечать потребностям обработки данных конкретной предметной области. Для других вариантов соединения используются внешние соединения, которые являются более сложными и позволяют получить более адекватные потребностям различных предметных областей данные.

Смешивание явных и неявных соединений не рекомендуется, но позволяется. Некоторые виды смешивания запрещены в Ред базе данных. В настоящее время синтаксис неявных соединений не рекомендуется к использованию.

Во многих случаях внутреннее соединение является подходящим средством, чтобы получить необходимые пользователю данные. Существует более гибкая система соединения таблиц — внешние соединения.

## Внешние соединения

Внешние соединения (OUTER JOIN) бывают левыми (LEFT), правыми (RIGHT) и полными (FULL). Внешнее соединение является очень гибким средством получения результирующего набора данных и предоставляет достаточно большие возможности для выполнения соединения многих различных таблиц, полученных из обычного оператора SELECT, представления или хранимой процедуры выбора.

**Левое внешнее соединение**

При левом внешнем соединении (LEFT OUTER JOIN) к столбцам строк первой, главной, левой, таблицы добавляются данные (дополнительные столбцы) из второй, правой, присоединяемой, таблицы на основании условий соединения, заданных в предложении ON. Если во второй, правой, таблице нет подходящей условию соединения строки, то соответствующий столбец первой соединяемой таблицы будет иметь пустое значение NULL. Общее количество строк, попадающих в результирующий набор данных, определяется условием в предложении WHERE.

**Примеры.** Пусть при выборе сотрудников организаций из таблицы STAFF для каждой строки нужно добавить на основании кода человека отсутствующие в записи фамилию, имя и отчество сотрудника, которые выбираются из таблицы людей PEOPLE. В этом случае можно использовать левое внешнее соединение таблицы STAFF с таблицей PEOPLE для получения соответствующих данных. Следующий оператор выполняет необходимую выборку:

**Пример 8.11**

```
SELECT
   PEOPLE.NAME3 || ' ' || PEOPLE.NAME1 || ' ' || PEOPLE.NAME2 AS "Сотрудник" ,
   DUTIES AS "Должность" ,
   SALARY AS "Оклад"
FROM STAFF
   LEFT OUTER JOIN PEOPLE
     ON STAFF.CODPEOPLE = PEOPLE.COD
ORDER BY 1;
```

Поскольку в операторе отсутствует предложение WHERE, в результирующий набор данных попадут все строки таблицы STAFF. Условие соединения задается предложением ON. Требуется естественное равенство кодов людей в таблице STAFF и в таблице PEOPLE. В строку результирующего набора данных будет добавлен один строковый столбец, который является конкатенацией фамилии, имени и отчества соответствующего человека. Если же в таблице людей нет соответствующих сведений о человеке (в данном случае это возможно только тогда, когда внешний ключ таблицы STAFF, столбец CODPEOPLE, имеет значение NULL), то этот столбец будет иметь пустое значение. Для выбора сотруд-

ников только одной конкретной организации нужно использовать предложение WHERE, в котором в качестве условия указать, например, код требуемой организации.

Посмотрите на оператор, где приведен пример внутреннего соединения (Пример 8.5). Результаты выполнения внутреннего и внешнего соединений различаются только тем, что набор данных, полученный при выполнении внутреннего соединения, будет содержать лишь те строки, для которых существует точное соответствие условию соединения. Строки таблицы STAFF, для которых нет соответствующих строк в таблице PEOPLE, не попадают в результирующий набор данных. В случае же внешнего левого соединения результирующий набор данных будет содержать, как правило, большее количество строк, потому что в нем будут присутствовать и строки из исходной таблицы STAFF, для которых отсутствуют соответствующие строки таблицы PEOPLE. Такое поведение системы во многих случаях при решении задач предметной области будет более естественным, поскольку в выходной набор данных попадают все строки сотрудников, даже если для них не существует соответствующих данных из таблицы людей, что может оказаться сигналом для людей, использующих систему, о том, что база данных не совсем адекватна текущей задаче обработке данных.

В следующем операторе выполняется двойное левое внешнее соединение трех таблиц — таблицы персонала STAFF, таблицы организаций FIRM и таблицы людей PEOPLE. Этот вариант является наиболее полным для получения всех необходимых данных о сотрудниках и соответствующих организациях.

**Пример 8.12**

```
SELECT
   P.FULLNAME AS "Сотрудник" ,
   S.DUTIES AS "Должность" ,
   F.NAME AS "Организация"
FROM STAFF S
   LEFT OUTER JOIN PEOPLE P
     ON S.CODPEOPLE = P.COD
   LEFT OUTER JOIN FIRM F
     ON F.COD = S.CODORG
ORDER BY 1;
```

Для каждой строки таблицы персонала выбирается соответствующая строка человека и строка необходимой организации, из которых в набор данных добавляются, соответственно, фамилия, имя, отчество из таблицы PEOPLE и полное название организации из таблицы FIRM.

Сравните этот оператор с выборкой данных при использовании явного и неявного внутреннего соединения, как показано в предыдущих примерах (Пример 8.8 и Пример 8.9). В данном варианте представлены более подробные данные обо всех сотрудниках, независимо от того, существуют ли соответствующие данные в таблице о людях и об организациях.

В следующем примере выполняется внешнее соединение таблицы с самой собой. Посмотрите листинг, где описана таблица людей PEOPLE (Пример 8.3). В таблице присутствуют столбцы, являющиеся внешними ключами, ссылающимися на ту же самую таблицу PEOPLE. Эти внешние ключи являются ссылками на отца (CODFATHER) и на мать (CODMOTHER) человека. Чтобы выбрать список всех людей и фамилии их отцов и матерей, следует выполнить оператор, как показано в следующем листинге.

**Пример 8.13**

```
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;
```

В результате будут выбраны все строки таблицы людей, независимо от того, есть ли для них соответствующие строки, заданные условиями соединения, то есть, существуют ли для них нужные сведения об отце и о матери.

Посмотрите пример 8.10, где выполнялось похожее действие с использованием внутреннего соединения. При внутреннем соединении в набор данных попадают лишь те строки, для которых найдены коды, указанные в условиях внутреннего соединения, то есть только для тех людей, у которых есть сведения об отце и о матери. Другие же люди в выборку не попадают.

Чтобы при использовании внешнего соединения получить такой же результат, как и во внутреннем соединении, необходимо лишь выполнить дополнительную проверку на отсутствие пустых значений NULL для кодов отца и матери в предложении WHERE. Соответствующий оператор выборки:

**Пример 8.14**

```
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
WHERE
   PG.CODMOTHER IS NOT NULL AND
   PG.CODFATHER IS NOT NULL
ORDER BY PG.FULLNAME;
```

**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.

Посмотрите пример 8.13, где выполняется двойное левое внешнее соединение таблицы с самой с собой. Чтобы получить такой же результат с использованием правого внешнего соединения, нужно выполнить оператор, в котором таблицы лишь переставлены в другом порядке:

**Пример 8.15**

```
SELECT
   PG.FULLNAME AS "Фамилия, имя, отчество" ,
   PM.NAME3 AS "Мать" ,
   PF.NAME3 AS "Отец"
FROM PEOPLE PF /* Главная таблица */
   RIGHT OUTER JOIN PEOPLE PM /* Мать */
     ON PF.CODMOTHER = PM.COD
   RIGHT OUTER JOIN PEOPLE PG /* Отец */
     ON PG.CODFATHER = PF.COD
ORDER BY PG.FULLNAME;
```

Здесь по сравнению с предыдущим листингом соединяемые таблицы просто записаны в обратном порядке.

**Полное внешнее соединение**

В случае полного внешнего соединения (FULL OUTER JOIN) выбираются все соответствующие условию в предложении WHERE строки как в левой, так и в правой таблице. Затем между этими строками устанавливается соответствие, заданное в предложении ON.

Посмотрите пример 8.11, где выполняется левое внешнее соединение таблицы персонала с таблицей людей для получения фамилий, имен и отчеств. В результате выполнения этого оператора полученный набор данных будет содержать все записи сотрудников, хранящиеся в базе данных с их фамилиями, именами и отчествами. Если для сотрудника будет найдено соответствие в таблице людей, то в запись будут помещены фамилия, имя и отчество этого человека, иначе в результирующем наборе данных соответствующие поля будут иметь пустое значение NULL. Можно выполнить полное внешнее соединение тех же таблиц сотрудников и людей (см. Пример 8.16). В отличие от примера 8.11 результирующий набор данных будет содержать все строки из таблицы персонала STAFF и все строки из таблицы людей PEOPLE. Там, где удовлетворяется условие соединения, заданное в предложении ON, в одну строку набора данных будут помещены соответствующие значения из двух таблиц. Иначе несоответствующие условиям поля будут иметь пустое значение NULL. В полном внешнем соединении количество строк результирующего набора данных будет больше.

**Пример 8.16**

```
SELECT
   PEOPLE.NAME3 || ' ' || PEOPLE.NAME1 || ' ' || PEOPLE.NAME2 AS "Сотрудник" ,
   DUTIES AS "Должность" ,
   SALARY AS "Оклад"
FROM STAFF
   FULL OUTER JOIN PEOPLE
     ON STAFF.CODPEOPLE = PEOPLE.COD
ORDER BY 1;
```

## Перекрестное соединение

Перекрестное соединение (CROSS) дает декартово произведение двух множеств строк обеих соединяемых таблиц. Каждая строка левой таблицы соединяется с каждой строкой правой таблицы.

Следующие три конструкции эквивалентны.

```
FROM <таблица1> CROSS JOIN <таблица2>
FROM <таблица1>, <таблица2>
FROM <таблица1> INNER JOIN <таблица2> ON <всегда истинное условие>
```

## 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.

**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. 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.

## 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.

**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.

## Предложение WHERE

В предложении WHERE задаются условия, которым должны удовлетворять строки исходных таблиц (представлений, таблиц, полученных из хранимых процедур выбора), перечисленных в предложении FROM, для того, чтобы они попали в результат выборки — результирующий набор данных.

### Синтаксис условия выборки данных

Условие выборки во многом по синтаксису похоже на условие домена, условие столбца таблицы или условие таблицы (листинг 8.8):

**Листинг 8.8. Синтаксис условий выборки в операторе SELECT**

```
<условие выборки> ::= {
    <значение> <оператор сравнения> { <значение 1> | (<выбор одного>) }
  | <значение> [NOT] IN ({ <значение 1> [, <значение 2>]... | <поиск одного> })
  | <значение> [NOT] BETWEEN <значение 1> AND <значение 2>
  | <значение> [NOT] LIKE <значение 1> [ESCAPE '<символ>']
  | <значение> IS [NOT] NULL
  | <значение> IS [NOT] DISTINCT FROM <значение 1>
  | <значение> <оператор сравнения> { ALL | SOME | ANY} ( <поиск одного> )
  | EXISTS (<поиск многих>)
  | SINGULAR (<поиск многих>)
  | <значение> [NOT] CONTAINING <значение 1>
  | <значение> [NOT] STARTING [WITH] <значение 1>
  | (<условие выборки>)
  | NOT <условие выборки>
  | <условие выборки> OR <условие выборки>
  | <условие выборки> AND <условие выборки> }
```

### Оператор сравнения

Оператором в этом условии является оператор сравнения (см. листинг 8.9).

**Листинг 8.9. Синтаксис оператора 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 8.1

Таблица 8.1 — Операторы сравнения

| **Операторы** | = | <> , != , ^= | > | < | >= , !< , ^< | <= , !> , ^> |
|---|---|---|---|---|---|---|
| **Значение** | Равно | Не равно | Больше | Меньше | Больше или равно, не меньше | Меньше или равно, не больше |

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.

## Значение в условии выборки данных

Термин «значение» в синтаксисе условия выборки данных определяется следующим образом:

**Листинг 8.10. Синтаксис значения в операторе выборки данных SELECT**

```
<значение> ::= {
 [{ <имя таблицы> | <псевдоним таблицы> }.]<имя столбца> [[<элемент массива>]]
  | <литерал>
  | <выражение>
  | NEXT VALUE FOR <имя генератора>
  | <обычная внутренняя функция> (<параметры>)
  | <агрегатная функция в операторе SELECT>
  | <функция UDF> [(<параметр> [, <параметр>]...)]
  | NULL }
```

Здесь можно указать имя столбца таблицы. Если несколько таблиц в операторе SELECT имеют столбцы с одинаковыми именами, то следует указать перед именем столбца имя или псевдоним таблицы и точку, чтобы избежать ненужной двусмысленности. Если для таблицы в операторе задан псевдоним, то можно указывать только псевдоним, но не имя таблицы. Иначе это приведет к ошибке.

Если столбец является массивом, то в квадратных скобках нужно указать и конкретный элемент массива. Если массив одномерный, то указывается номер этого элемента (с учетом заданного диапазона значений для элементов массива). Для многомерных массивов нужно указать номера позиций каждого из диапазонов, разделяя их запятыми.

*Литерал* — это числовая константа, строковая константа, заключенная в апострофы, литерал даты или времени, предварительно определенный литерал, контекстная переменная (см. главу 2 «Типы данных Ред База Данных»).

Существует четыре предварительно определенных литерала даты:

- 'NOW' типа TIMESTAMP возвращает текущую дату и текущее время;

- 'TODAY' типа DATE возвращает текущую дату;
- 'TOMORROW' типа DATE возвращает завтрашнюю дату;
- 'YESTERDAY' типа DATE возвращает вчерашнюю дату.

Эти литералы не чувствительны к регистру.

Контекстными переменными, которыми можно пользоваться при создании таблицы, являются:

- CURRENT_TIMESTAMP типа TIMESTAMP возвращает текущую дату и текущее время.
- CURRENT_DATE типа DATE возвращает текущую дату.
- CURRENT_TIME типа TIME возвращает текущее время.
- Контекстная переменная CURRENT_CONNECTION возвращает число — системный идентификатор текущего соединения с базой данных.
- CURRENT_USER возвращает имя пользователя, который соединен с базой данных.
- USER — имя пользователя, связанного с текущим экземпляром клиентской библиотеки. Полностью соответствует значению, полученному из контекстной переменной CURRENT_USER.
- Контекстная переменная CURRENT_ROLE возвращает имя роли, под которой с базой данных соединился пользователь.
- CURRENT_TRANSACTION возвращает число — системный идентификатор транзакции, под управлением которой выполняется текущий запрос.

Подробные описания характеристик и порядка использования предварительно определенных литералов и контекстных переменных см. в главе 2 «Типы данных Ред База Данных».

*Выражением* может быть сколь угодно сложное правильное выражение SQL, содержащее допустимые операции, обращения к литералам, функциям.

Конструкция NEXT VALUE FOR позволяет получить новое, следующее значение генератора. Обычно используется для формирования значения искусственного первичного ключа.

## Использование встроенных функций

В SQL СУБД Ред База Данных существует два типа встроенных функций — обычные встроенные функции и агрегатные функции в операторе SELECT.

*Обычная встроенная функция* — это функция, работающая с одним или более параметрами, которая напрямую не связана с оператором SQL выборки данных SELECT. Функция возвращает ровно одно значение. Параметры передаются таким функциям на основании принятого для каждой функции синтаксиса. Агрегатные функции в операторе SELECT — функции, определенные в языке SQL Ред База Данных. Они работают не с одним фиксированным набором параметров, а с группой значений, полученных при выполнении определенного оператора SELECT из таблицы базы данных. Агрегатные функции используются внутри списка выбора этого оператора SELECT.

Описание всех функций см. в приложении E «Функции». Подробные описания агрегатных функций см. также в главе 4 «Работа с доменами».

**Пример 1.** Пусть имеется таблица STAFF, описывающая персонал организации. Чтобы подсчитать количество сотрудников можно использовать агрегатную функцию COUNT:

```
SELECT COUNT (*)
FROM STAFF;
```

**Пример 2.** Для определения месячного фонда заработной платы сотрудников в таблице STAFF нужно просуммировать оклады всех сотрудников — столбец SALARY:

```
SELECT SUM (SALARY)
FROM STAFF;
```

**Пример 3.** Для вычисления средней заработной платы сотрудников в таблице STAFF нужно использовать функцию AVG:

```
SELECT AVG (SALARY)
FROM STAFF;
```

В следующем операторе из таблицы сотрудников STAFF выбираются только те сотрудники, чей оклад меньше среднего значения оклада:

**Пример 8.17**

```
SELECT *
FROM STAFF
WHERE SALARY < (SELECT AVG (SALARY) FROM STAFF);
```

Значением в предложении WHERE также может быть и обращение к функции, определенной пользователем (User Defined Function, UDF — см. приложение Г).

В качестве значения может быть указано пустое значение NULL или ключевое слово USER, означающее имя пользователя, соединенного в настоящий момент с базой данных. Не следует NULL включать в какую-либо операцию сравнения. Результатом всегда будет неопределенное значение UNKNOWN. Лучше использовать конструкции IS NULL и IS NOT NULL.

## Операторы SELECT, используемые в условии выборки данных

*Выбор одного* — это оператор SELECT, возвращающий в точности одно значение одного столбца. Пустое значение недопустимо.

*Поиск одного* — оператор SELECT, возвращающий произвольное количество значений одного столбца. Здесь возможно и пустое значение NULL.

*Поиск многих* — оператор SELECT, возвращающий ноль или произвольное количество значений нескольких столбцов.

## Оператор IN

Оператор IN указывает, что значение в столбце выбираемой таблицы должно находиться (или не находиться, если указано ключевое слово NOT) в заданном списке.

Список может быть представлен в виде действительно списка явно указанных значений (литералов) или этот список может быть получен при выполнении определенного оператора SELECT. Весь список заключается в круглые скобки. Отдельные значения должны разделяться запятыми.

Список также можно задать и с использованием оператора SELECT, который выбирает произвольное количество значений ровно одного столбца из таблицы (таблиц) базы данных.

Символьные данные в операторе чувствительны к регистру.

```
<значение > [NOT] IN ({ <значение 1> [, <значение 2>]... | <поиск одного> })
```

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

В операторе неявно допускается и пустое значение NULL.

Если нужно убрать чувствительность к регистру, то следует к значению в левой части этого выражения применить функцию UPPER, а значения в списке записывать прописными буквами. Либо использовать функцию LOWER, записывая значения строчными буквами.

## Оператор BETWEEN

В этом операторе проверяется наличие значения, записанного в левой части условия, в диапазоне, заданном в правой части условия, включая граничные значения. Начальное значение должно быть не больше конечного значения в диапазоне.

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

Условие будет истинным, если значение присутствует в указанном диапазоне (от `<значение 1>` 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>)
```

**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 *
FROM STAFF
WHERE SALARY BETWEEN 10000 AND 20000;
```

Здесь выбираются сотрудники, чьи оклады находятся в диапазоне 10000 и 20000 включительно.

## Оператор LIKE

Этот оператор задает проверку наличия (или отсутствия в случае указания необязательного ключевого слова `NOT`) в значении столбца символьного типа данных определенных символов, заданных в этом операторе.

```
<значение> [NOT] LIKE <значение 1> [ESCAPE '<один символ>']
```

Этот вариант является чувствительным к регистру. В исходной строке можно указать шаблонные символы `%` и `_`. Символ процента задает произвольное количество, в том числе и нулевое, любых символов. Знак подчеркивания задает ровно один любой символ.

Чтобы этот вариант можно было применять как к строчным, так и к прописным буквам, следует использовать функцию перевода букв в прописные — `UPPER` для левой части условия:

```
UPPER(<значение>) LIKE '<строка, состоящая только из прописных букв>'
```

Можно также использовать и внутреннюю функцию `LOWER`, переводящую все буквы в нижний регистр.

Необязательное ключевое слово `ESCAPE` позволяет в строку поиска включить и сами шаблонные символы `%` и `_`. Здесь нужно указать символ, который должен предшествовать в строке поиска шаблонному символу, когда такой шаблонный символ должен рассматриваться как обычный символ, присутствующий в строке.

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.

## Оператор IS NULL

Этот оператор осуществляет проверку помещаемого в столбец значение на пустое значение `NULL` (или отсутствие пустого значения в случае присутствия ключевого слова `NOT`). Оператор может вернуть только истинное значение `TRUE` или ложное `FALSE`, значение `UNKNOWN` невозможно.

```
<значение> IS [NOT] NULL
```

## Оператор IS DISTINCT FROM

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

## Функции ALL, SOME, ANY

Синтаксис использования этих функций:

```
<значение> <оператор сравнения> { ALL | SOME | ANY } (<поиск одного>)
```

Здесь используется операторы сравнения, описанные выше в этом разделе, а также оператор IS [NOT] DISTINCT FROM. Аргументом любой из функций является оператор SELECT, возвращающий произвольное количество значений одного столбца. Допустимо также и пустое значение.

Функция ALL вернет значение «истина», если сравнение будет истинным для всех значений столбца, полученных из оператора SELECT.

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

## Функция EXISTS

Синтаксис этой функции:

```
EXISTS (<поиск многих>)
```

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

Результатом будет «истина», если оператор SELECT вернет хотя бы одно значение, соответствующее условиям поиска, заданным в предложении WHERE.

## Функция SINGULAR

Синтаксис функции:

```
SINGULAR (<поиск многих>)
```

Аргументов функции SINGULAR является оператор SELECT, возвращающий произвольное количество любых столбцов таблицы (обычно это *).

Результатом будет «истина», если оператор SELECT, заданный в конструкции «поиск многих», вернет в точности одно значение, соответствующее условиям поиска, заданным в предложении WHERE.

## Оператор CONTAINING

Синтаксис этого оператора:

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

Результатом будет «истина», если значение в левой части выражения будет содержать в качестве своей части значение, указанное в правой части. Этот оператор не чувствителен к регистру. К нему нет необходимости применять функцию UPPER.

### Оператор STARTING WITH

Синтаксис этого оператора:

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

Результатом будет «истина», если значение в левой части выражения будет начинаться с символов, указанных в правой части. Оператор чувствителен к регистру, однако такое ограничение также можно легко обойти, используя функцию UPPER или функцию LOWER.

### Логические операции с условиями выборки

При описании условий выборки данных в операторе SELECT можно использовать круглые скобки, чтобы задать высший приоритет выполнения проверки части условия. Можно использовать логические операции отрицания (NOT), дизъюнкции (OR) и конъюнкции (AND).

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). Любое сравнение, где одним из значений является пустое значение NULL, дает результат UNKNOWN. Следующие таблицы, называемые таблицами истинности, дают точное определение логическим операциям отрицания, дизъюнкции и конъюнкции.

В операции отрицания присутствует один операнд. Результат выполнения отрицания представлен в таблице 8.2.

Таблица 8.2 — Операция отрицания NOT

| Операнд | NOT операнд |
|---|---|
| TRUE | FALSE |
| FALSE | TRUE |
| UNKNOWN | UNKNOWN |

В операции дизъюнкции (логическое ИЛИ) участвуют два операнда. Результат выполнения дизъюнкции показан в таблице 8.3.

Таблица 8.3 — Операция дизъюнкции OR

| Операнд 1 | Операнд 2 | Операнд 1 OR Операнд 2 |
|---|---|---|
| TRUE | TRUE | TRUE |
| TRUE | FALSE | TRUE |
| FALSE | TRUE | TRUE |
| FALSE | FALSE | FALSE |
| TRUE | UNKNOWN | TRUE |
| FALSE | UNKNOWN | UNKNOWN |
| UNKNOWN | TRUE | TRUE |
| UNKNOWN | FALSE | UNKNOWN |
| UNKNOWN | UNKNOWN | UNKNOWN |

В операции конъюнкции (логическое И) участвуют два операнда. Результат выполнения конъ-

юнкции показан в таблице 8.4.

Таблица 8.4 — Операция конъюнкции AND

| Операнд 1 | Операнд 2 | Операнд 1 AND Операнд 2 |
|---|---|---|
| TRUE | TRUE | TRUE |
| TRUE | FALSE | FALSE |
| FALSE | TRUE | FALSE |
| FALSE | FALSE | FALSE |
| TRUE | UNKNOWN | UNKNOWN |
| FALSE | UNKNOWN | FALSE |
| UNKNOWN | TRUE | UNKNOWN |
| UNKNOWN | FALSE | FALSE |
| UNKNOWN | UNKNOWN | UNKNOWN |

Порядок выполнения операций следующий:

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

Операции с одинаковым приоритетом выполняются слева направо.

## Предложения GROUP BY и HAVING

Необязательное предложение GROUP BY задает условие группирования выбранных данных в соответствии со значением указанного столбца (указанных столбцов). 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. В предложении GROUP BY можно использовать имена столбцов, их псевдонимы или номера столбцов в заданном в операторе SELECT списке выбора. Синтаксис предложения представлен в листинге 8.11.

**Листинг 8.11. Синтаксис предложения GROUP BY**

```
GROUP BY {
    <имя столбца из списка выбора>
  | <псевдоним столбца из списка выбора>
  | <номер столбца в списке выбора>
  | <не агрегатное выражение, не включено в список выборки>}
  [, { <имя столбца из списка выбора>
     | <псевдоним столбца из списка выбора>
     | <номер столбца в списке выбора>
```

```
    | <не агрегатное выражение, не включено в список выборки>} ...]
[HAVING <условие выборки>]
```

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.

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).

В параметры предложения GROUP BY могут быть включены не только имена, псевдонимы и номера столбцов из списка выборки, но и следующее:

- 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, то невозможно сказать какому значению столбца соответствует значение агрегированной строки.
- 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.

**Пример.** Следующий оператор использует четыре агрегатные функции для расчета среднего значения заработной платы, поиска минимального и максимального значения заработной платы, подсчета количества сотрудников. Все это выполняется для каждой организации, хранящейся в базе данных. Данные выбираются из таблицы сотрудников STAFF.

**Пример 8.18**

```
SELECT
   AVG(SALARY) AS "Среднее" ,
   MAX(SALARY) AS "Максимум ,
   MIN(SALARY) AS "Минимум" ,
   COUNT(*) AS "Количество" ,
   S.CODORG AS "Организация"
FROM STAFF S
GROUP BY "Организация"
ORDER BY 4;
```

В предложении GROUP BY группировка задается по единственному столбцу из списка выбора, не входящему в состав параметров агрегатных функций — по коду организации, где работает человек. Обратите внимание — в предложении GROUP BY использован псевдоним этого столбца, что допустимо в настоящей версии Ред База Данных.

Предложение HAVING позволяет задать условие, на основании которого сгруппированные строки попадут в результирующий набор данных. Общее ограничение количества выбираемых строк задается предложением WHERE. Предложение HAVING позволяет задать дополнительное ограничение строк в множестве строк, уже сгруппированных на основании предложения GROUP BY. В следующем примере осуществляется такая же выборка данных, что и в предыдущем примере, но только по тем организациям, где минимальное значение оклада превышает 1500:

**Пример 8.19**

```
SELECT
   AVG(SALARY) AS "Среднее" ,
   MAX(SALARY) AS "Максимум ,
   MIN(SALARY) AS "Минимум" ,
   COUNT(*) AS "Количество" ,
   S.CODORG AS "Организация"
FROM STAFF S
GROUP BY "Организация"
   HAVING MIN(SALARY) > 1500
ORDER BY 4;
```

Группировку полученных данных можно использовать в любом, сколь угодно сложном операторе выборки данных, содержащем агрегатные функции. Следующий пример 8.20 расширяет предыдущий оператор выборки и содержит левое внешнее соединение таблицы сотрудников `STAFF` с таблицей организаций `FIRM`. Соединение используется для помещения в результирующий набор данных не малоинформативного для пользователя системы кода организации, а названия этой организации.

**Пример 8.20**

```
SELECT
   AVG(SALARY) AS "Среднее" ,
   MAX(SALARY) AS "Максимум ,
   MIN(SALARY) AS "Минимум" ,
   COUNT(*) AS "Количество" ,
   F.NAME AS "Организация"
FROM STAFF S
   LEFT OUTER JOIN FIRM F
     ON F.COD = S.CODORG
GROUP BY "Организация"
   HAVING MIN(SALARY) > 1500
ORDER BY 4;
```

Подробнее о видах соединения таблиц см. в разделе «Соединение таблиц (JOIN)» в этой главе.

## Предложение UNION

Предложение `UNION` позволяет объединить выбранный основным оператором `SELECT` набор данных с другим набором данных, имеющим точно такую же структуру. Не следует путать объединение таблиц, `UNION`, с соединением таблиц, `JOIN`. Синтаксис предложения объединения `UNION` следующий (см. листинг 8.12):

**Листинг 8.12. Синтаксис предложения `UNION`**

```
UNION [DISTINCT | ALL] <другой набор данных>
```

Присоединяемый (объединяемый) набор данных должен иметь тот же состав столбцов по количеству и типам данных, что и основной набор данных, полученный главным оператором `SELECT`. Допустимо лишь отличие в количестве символов у строковых типов данных.

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.

Другой набор данных, указанный в синтаксисе оператора `UNION`, — это оператор `SELECT`, ко-

торый обращается к другой или той же самой объединяемой таблице, представлению или хранимой процедуре выбора.

В одном операторе SELECT может присутствовать более одного объединения.

**Пример объединения.** Пусть в базе данных присутствуют две таблицы, одна из которых описывает административные правонарушения людей, а другая — награды, присужденные людям.

**Пример 8.21**

```
/*** Правонарушения людей ***/
CREATE TABLE PEOPLEADMIN (
   COD INTEGER NOT NULL, /* Код - первичный ключ */
   CODPEOPLE INTEGER, /* Код человека */
   CODADMIN CHAR(8), /* Код правонарушения */
   DATEADMIN DATE, /* Дата правонарушения */
   CONSTRAINT PK_PEOPLEADMIN PRIMARY KEY (COD),
   CONSTRAINT FK1_PEOPLEADMIN
     FOREIGN KEY (CODPEOPLE) REFERENCES PEOPLE (COD)
       ON DELETE CASCADE,
   CONSTRAINT FK2_PEOPLEADMIN
     FOREIGN KEY (CODADMIN) REFERENCES REFADMIN (COD)
       ON UPDATE CASCADE
       ON DELETE CASCADE );
CREATE SEQUENCE GEN_PEOPLEADMIN;
/*** Награды людей ***/
CREATE TABLE PEOPLEREW (
   COD INTEGER NOT NULL, /* Код - первичный ключ */
   CODPEOPLE INTEGER, /* Код человека */
   CODREW CHAR(6), /* Код награды */
   DATESPEC DATE, /* Дата получения */
   CONSTRAINT PK_PEOPLEREW PRIMARY KEY (COD),
   CONSTRAINT FK1_PEOPLEREW
     FOREIGN KEY (CODPEOPLE) REFERENCES PEOPLE (COD)
       ON DELETE CASCADE,
   CONSTRAINT FK2_PEOPLEREW
     FOREIGN KEY (CODREW) REFERENCES REFREWARD (COD)
       ON UPDATE CASCADE
       ON DELETE CASCADE );
CREATE SEQUENCE GEN_PEOPLEREW;
```

Можно объединить эти две таблицы, если это требуется для решения задачи предметной области. В состав столбцов объединенного набора данных даже можно включить столбцы «код правонарушения» и «код награды», хотя у них и не совпадают размеры, но тип данных одинаковый — символьный.

Возможный вариант объединения этих двух таблиц:

**Пример 8.22**

```
SELECT
   COD AS "Внутренний код" ,
   CODPEOPLE AS "Код человека" ,
   DATEADMIN AS "Дата" ,
   'Нарушение'AS "Вид"
FROM PEOPLEADMIN
UNION
```

```
   SELECT
     COD AS "Внутренний код
     CODPEOPLE AS "Код человека
     DATESPEC AS "Дата
     'Награда'AS "Вид"
   FROM PEOPLEREW
ORDER BY 1;
```

Четвертым столбцом в списке выбора указывается строковая константа, которая определяет, является ли выводимая строка сведением об административном нарушении («Нарушение») или о награде («Награда»). Псевдонимом для этого столбца выбрано слово «Вид». Этот же текст будет помещен в заголовок отображения таблицы.

Таблицу можно объединять и саму с собой. В следующем примере показано двойное объединение таблиц. Здесь в список столбцов включены и столбцы строкового типа данных различной размерности — CODADMIN и CODREW.

**Пример 8.23**

```
SELECT
   COD AS "Внутренний код" ,
   CODPEOPLE AS "Код человека" ,
   DATEADMIN AS "Дата" ,
   CODADMIN AS "Код нарушения/награды"
FROM PEOPLEADMIN
UNION
   SELECT
     COD AS "Внутренний код" ,
     CODPEOPLE AS "Код человека" ,
     DATESPEC AS "Дата" ,
     CODREW AS "Код нарушения/награды"
   FROM PEOPLEREW
UNION ALL
   SELECT
     COD AS "Внутренний код" ,
     CODPEOPLE AS "Код человека" ,
     DATEADMIN AS "Дата" ,
     CODADMIN AS "Код нарушения/награды"
   FROM PEOPLEADMIN
ORDER BY 1;
```

Во втором предложении UNION выполняется объединение с той же самой таблицей. Это так называемое реентерабельное объединение. Чтобы в результирующем наборе данных действительно получить все дублирующие строки таблицы правонарушений людей PEOPLEADMIN, необходимо в предложении UNION указать и ключевое слово ALL. Иначе в выходной набор данных не попадут дубликаты уже выбранных из таблицы строк.

Операторы SELECT, как главный, так и присутствующие в предложении UNION, могут быть произвольной сложности. В таких операторах могут присутствовать и предложения PLAN — см. раздел «Предложение PLAN» этой главы.

Например, в предыдущий оператор SELECT можно внести добавление — выполнить внутреннее соединение (INNER JOIN) с таблицей людей для получения фамилии, имени и отчества каждого человека. Соединения нужно выполнять как для главного оператора SELECT, так и для всех операторов выборки данных в последующих предложениях UNION, чтобы структура объединяемых наборов данных было одинаковой. Усложненный вариант выборки представлен в Примере 8.24.

**Пример 8.24**

```
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 1;
```

Вместо внутреннего соединения здесь также для получения того же результата можно выполнить и левое внешнее соединение. Подробнее о соединениях см. в разделе «Соединение таблиц (JOIN)» в этой главе.

Однако в такой форме записи объединения в предложении `ORDER BY` нельзя использовать алиасы столбцов. Этот оператор можно также записать и в несколько ином более правильном виде:

**Пример 8.25**

```
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";
```

Оператор SELECT верхнего уровня выбирает все столбцы из таблицы, которая получается в результате объединения трех таблиц. Операция выборки объединенных и соединенных таблиц записывается после ключевого слова FROM главного оператора SELECT и заключается в круглые скобки.

В таком варианте в предложении ORDER BY можно использовать и псевдонимы столбцов, по которым выполняется упорядочение.

## Предложение 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 8.13.

**Листинг 8.13. Синтаксис предложения PLAN в операторе SELECT**

```
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> ...])

<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.

Первый самый простой запрос. Выбираются все столбцы всех строк таблицы стран COUNTRY:

```
SELECT *
FROM COUNTRY;
```

Оптимизатор построит следующий план выборки данных:

```
PLAN (COUNTRY NATURAL)
```

В расширенном варианте (команда SET EXPLAIN) план выглядит так:

```
Select Expression
   -> Table "COUNTRY" Full Scan
```

По этому плану все строки таблицы просматриваются последовательно. Точно такой же результат можно получить, если добавить предложение плана в оператор SELECT:

```
SELECT *
FROM COUNTRY
PLAN (COUNTRY NATURAL);
```

В следующем запросе задается еще и упорядоченность по столбцу, являющемуся первичным ключом таблицы.

```
SELECT *
FROM COUNTRY
```

```
ORDER BY CODCOUNTRY;
```

Оптимизатор построит следующий план:

```
PLAN (COUNTRY ORDER PK_COUNTRY)
```

В расширенном варианте план выглядит так:

```
Select Expression
   -> Table "COUNTRY"Access By ID
       -> Index "PK_COUNTRY" Full Scan
```

В элементе плана задается упорядочение строк промежуточного набора данных с использованием индекса PK_COUNTRY, построенного системой автоматически для поддержания значений первичного ключа таблицы COUNTRY.

Эквивалентным этому запросу будет запрос, содержащий такой план:

```
SELECT *
FROM COUNTRY
PLAN SORT (COUNTRY INDEX (PK_COUNTRY))
ORDER BY CODCOUNTRY;
```

Для запроса, содержащего упорядочение по столбцу, не являющемуся ключевым и не входящему в состав какого-нибудь индекса

```
SELECT *
FROM COUNTRY
ORDER BY NAME;
```

будет построен план, в котором сразу после ключевого слова PLAN указывается упорядочение полученного набора данных:

```
PLAN SORT (COUNTRY NATURAL)
```

В расширенном варианте план выглядит так:

```
Select Expression
   -> Sort (record length: 148, key length: 36)
       -> Table "COUNTRY" 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.

В следующем запросе выбираются строки из таблицы регионов. Результат упорядочивается по столбцу «код страны», который входит в состав первичного ключа таблицы и в то же время является внешним ключом, ссылающимся на первичный ключ родительской таблицы — справочника стран.

```
SELECT *
FROM REGION
ORDER BY CODCOUNTRY;
```

План задает выборку строк в порядке, указанном в индексе, который был автоматически создан системой для внешнего ключа (FK_REGION):

```
PLAN (REGION ORDER FK_REGION)
```

В расширенном варианте план выглядит так:

```
Select Expression
   -> Table "REGION" Access By ID
       -> Index "FK_REGION" Full Scan
```

Оптимизатор, естественно, выбирает упорядочение отыскиваемых строк на основании внешнего ключа. В данном случае при задании плана можно в качестве индекса для упорядочения указать и индекс, созданный для первичного ключа. В состав первичного ключа входит код страны и код региона. Поскольку код страны является первым в структуре первичного ключа, такой индекс будет вполне приемлемым решением при выборке необходимых строк.

```
PLAN (REGION ORDER PK_REGION)
```

Если в операторе `SELECT` при выборке регионов в предложении `ORDER BY` указать столбцы, входящие в состав первичного ключа (код страны и код региона) в правильном порядке

```
SELECT *
FROM REGION
ORDER BY CODCOUNTRY, CODREGION;
```

то оптимизатор выберет упорядочение строк набора данных по первичному ключу:

```
PLAN (REGION ORDER PK_REGION)
```

В расширенном варианте план выглядит так:

```
Select Expression
   -> Table "REGION"Access By ID
       -> Index "PK_REGION"Full Scan
```

Однако если в списке упорядочения переставить местами столбцы или изменить направление упорядочения на убывающее, то в такой таблице не найдется подходящего индекса.

```
SELECT *
FROM REGION
ORDER BY CODREGION, CODCOUNTRY;
```

В этом случае план будет создан в следующем виде:

```
PLAN SORT (REGION NATURAL)
```

В расширенном варианте план выглядит так:

```
Select Expression
   -> Sort (record length: 36, key length: 16)
       -> Table "REGION"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 регионов 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.

В следующем запросе данные выбираются также из таблицы регионов, 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.

```
SELECT *
FROM REGION
WHERE CODCOUNTRY = 'USA';
```

Оптимизатор построит план, в котором используется индекс, автоматически построенный системой для ограничения внешнего ключа:

```
PLAN (REGION INDEX (FK_REGION))
```

В расширенном варианте план выглядит так:

```
Select Expression
   -> Filter
       -> Table "REGION" Access By ID
           -> Bitmap
               -> Index "FK_REGION" Range Scan (full match)
```

Использование индекса внешнего ключа в данном случае позволяет резко ускорить выборку релевантных данных.

В предыдущий запрос можно ввести требование упорядоченности результирующего набора данных по столбцам, входящим в состав первичного ключа.

```
SELECT * FROM REGION
WHERE CODCOUNTRY = 'USA'
ORDER BY CODCOUNTRY, CODREGION;
```

Созданный оптимизатором план теперь будет включать использование индекса первичного ключа для упорядочения строк набора данных на основании предложения ORDER BY:

```
PLAN (REGION ORDER PK_REGION)
```

В расширенном варианте план выглядит так:

```
Select Expression
   -> Filter
       -> Table "REGION"Access By ID
           -> Index "PK_REGION"Range Scan (partial match: 1/2)
```

## Примеры составных планов

При выполнении более сложного запроса, включающего объединения (UNION) трех таблиц, создается несколько планов.

**Пример 8.26**

```
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"
   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;
```

Оптимизатор создаст три плана для каждого оператора SELECT:

```
PLAN SORT (PEOPLEADMIN NATURAL, PEOPLEREW NATURAL, PEOPLEADMIN NATURAL)
```

В расширенном варианте план выглядит так:

```
Select Expression
   -> Unique Sort (record length: 104, 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).

Следующий оператор SELECT выполняет двойное левое внешнее соединение таблицы с этой же таблицей:

**Пример 8.27**

```
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;
```

На основании этого запроса оптимизатор создаст следующий план:

```
PLAN JOIN (SORT (JOIN (PG NATURAL, PM INDEX (PK_PEOPLE))), PF INDEX (PK_PEOPLE))
```

В расширенном варианте план выглядит так:

```
Select Expression
   -> Nested Loop Join (outer)
       -> Sort (record length: 190, key length: 60)
           -> Nested Loop Join (outer)
               -> Table "PEOPLE" as "PG" Full Scan
               -> Filter
                   -> Table "PEOPLE" as "PM" Access By ID
                       -> Bitmap
                           -> Index "PK_PEOPLE" Unique Scan
       -> Filter
           -> Table "PEOPLE" as "PF" Access By ID
               -> Bitmap
                   -> Index "PK_PEOPLE" 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.

**Пример 8.28**

```
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;
```

Оптимизатор создаст следующий план:

```
PLAN JOIN (JOIN (PG ORDER PK_PEOPLE, PM INDEX (PK_PEOPLE)), PF INDEX (PK_PEOPLE))
```

В расширенном варианте план выглядит так:

```
Select Expression
   -> Nested Loop Join (outer)
       -> Nested Loop Join (outer)
           -> Table "PEOPLE" as "PG" Access By ID
               -> Index "PK_PEOPLE" Full Scan
           -> Filter
               -> Table "PEOPLE" as "PM" Access By ID
                   -> Bitmap
                       -> Index "PK_PEOPLE" Unique Scan
       -> Filter
           -> Table "PEOPLE" as "PF" Access By ID
               -> Bitmap
                   -> Index "PK_PEOPLE" 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.

Выполнение довольно сложного запроса, который содержит как объединения (`UNION`), так и соединения (`JOIN`) нескольких таблиц, приведет к созданию также довольно сложного плана.

**Пример 8.29**

```
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";
```

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 главных таблиц выбираются последовательным перебором. Данные из соединяемых таблиц выбираются при

использовании индекса, построенного для первичного ключа таблицы людей (PK_PEOPLE):

```
PLAN SORT (SORT (JOIN (P NATURAL, PEOPLEADMIN INDEX (FK1_PEOPLEADMIN)),
                JOIN (P NATURAL, PEOPLEREW INDEX (FK1_PEOPLEREW))),
                JOIN (P NATURAL, PEOPLEADMIN INDEX (FK1_PEOPLEADMIN)))
```

В расширенном варианте план выглядит так:

```
Select Expression
   -> Sort (record length: 154, key length: 60)
       -> Union
           -> Unique Sort (record length: 148, key length: 84)
               -> Union
                   -> Nested Loop Join (inner)
                       -> Table "PEOPLE" as "P" Full Scan
                       -> Filter
                           -> Table "PEOPLEADMIN" Access By ID
                               -> Bitmap
                                   -> Index "FK1_PEOPLEADMIN" Range Scan(full match)
                   -> Nested Loop Join (inner)
                       -> Table "PEOPLE" as "P" Full Scan
                       -> Filter
                           -> Table "PEOPLEREW" Access By ID
                               -> Bitmap
                                   -> Index "FK1_PEOPLEREW" Range Scan (full match)
           -> Nested Loop Join (inner)
               -> Table "PEOPLE" as "P" Full Scan
               -> Filter
                   -> Table "PEOPLEADMIN" Access By ID
                       -> Bitmap
                           -> Index "FK1_PEOPLEADMIN" 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).

В следующем запросе присутствуют агрегатные функции, выполняется полное внешнее соединение и группировка результата.

**Пример 8.30**

```
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;
```

Оптимизатор построит следующий план:

```
PLAN SORT (SORT (JOIN (JOIN (O NATURAL, S INDEX (FK2_STAFF)),
                       JOIN (S NATURAL, O INDEX (PK_FIRM)))))
```

В расширенном варианте план выглядит так:

```
Select Expression
   -> Sort (record length: 180, key length: 64)
       -> Aggregate
           -> Sort (record length: 116, key length: 64)
               -> Full Outer Join
                   -> Nested Loop Join (outer)
                       -> Table "FIRM"as "O"Full Scan
                       -> Filter
                           -> Table "STAFF"as "S"Access By ID
                               -> Bitmap
                                   -> Index "FK2_STAFF"Range Scan (full match)
                   -> Nested Loop Join (anti)
                       -> Table "STAFF"as "S"Full Scan
                       -> Filter
                           -> Table "FIRM"as "O"Access By ID
                               -> Bitmap
                                   -> Index "PK_FIRM"Unique Scan
```

## Предложение ORDER BY

Результат выборки данных при выполнении оператора SELECT по определению 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 8.14:

**Листинг 8.14. Синтаксис предложения ORDER BY**

```
ORDER BY <упорядочиваемый элемент> [, <упорядочиваемый элемент> ... ]

<упорядочиваемый элемент> ::=
   {<имя столбца>|<псевдоним столбца>|<номер столбца>|<произвольное выражение>}
   [COLLATE <порядок сортировки>]
   [ASC[ENDING] | DESC[ENDING]]
   [NULLS {FIRST | LAST}]
```

В предложении через запятую перечисляются столбцы, по которым нужно упорядочить результирующий набор данных. Можно задавать имя столбца, псевдоним, присвоенный столбцу в списке выбора при помощи ключевого слова AS, или порядковый номер столбца в списке выбора. В одном предложении можно для разных столбцов смешивать форму записи. Например, один столбец в списке упорядочивания может быть задан своим именем, а другой порядковым номером.

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 позволяет задать порядок сортировки строкового столбца, если нужен порядок, отличный от того, который был установлен для этого столбца (явно при описании столбца или по умолчанию, принятому для соответствующего набора символов). Допустимые порядки сортировки для различных наборов символов см. в приложении В «Наборы символов и порядок

сортировки».

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.

В предложении ORDER BY формально можно задавать и столбцы, имеющие тип данных BLOB. Не рекомендуется использовать такую возможность, поскольку это дает неверные результаты.

## 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.

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 SORT (COUNTRY NATURAL)
```

В расширенном варианте план выглядит так:

```
Select Expression
   -> Sort (record length: 62, key length: 24)
       -> Table "COUNTRY" Full Scan
```

## Предложение ROWS

Предложение ROWS задает диапазон строк, которые попадут в результирующий набор данных из полученного в результате выполнения запроса набора данных. Синтаксис предложения:

```
ROWS <значение 1> [TO <значение 2>]
```

Предложение ROWS можно использовать, только если задано и предложение ORDER BY и не заданы предложения FIRST, SKIP и OFFSET, FETCH.

Значение 1 задает количество включаемых в выходной набор данных строк, упорядоченных в предложении ORDER BY, если не задан вариант TO. Это первые строки в упорядоченном по ORDER BY списке.

Значение 1 задает начальный номер строки в упорядоченном списке строк, если задан вариант TO. Значение 2 в этом случае указывает конечный номер строки.

Значением здесь также может быть число или выражение, возвращающее числовое значение. Если используется выражение, то оно должно быть заключено в круглые скобки. Если в выражении присутствует и оператор SELECT, то он дополнительно должен быть заключен в круглые скобки. Например, для выбора первой половины строк из упорядоченного списка, полученного из таблицы FIRM, можно использовать следующее предложение ROWS:

```
ROWS ((SELECT COUNT (*) FROM FIRM) / 2)
```

В этом синтаксисе значение 1 задает количество включаемых в результирующий набор данных строк, упорядоченных на основании предложения ORDER BY, если не задано ключевое слово TO. Это первые строки в упорядоченном наборе данных. Если набор данных содержит меньше, чем указано строк, то в результирующий набор данных будут помещены только существующее количество строк без выдачи каких-либо сообщений.

Если в предложении ROWS задано и ключевое слово TO, то в результирующий набор данных помещаются строки из упорядоченного набора, начиная с номера значение 1 и заканчивая номером значение 2 включительно. Если количество строк набора данных не соответствует указанным в предложении ROWS номерам строк, то в результирующий набор данных будут помещены только существующие строки без выдачи каких-либо сообщений.

### Соотношения между ключевыми словами FIRST и SKIP и предложением ROWS

В одном предложении SELECT не могут одновременно присутствовать ключевые слова FIRST и SKIP и в то же время предложение ROWS. В любом случае в операторе может присутствовать предложение WHERE. Между этими двумя вариантами отбора строк по их номерам существуют следующие соотношения.

Случай, когда в предложении ROWS не указано ключевое слово TO:

```
ROWS <значение 1>
```

Это в точности соответствует варианту задания только ключевого слова FIRST:

```
FIRST <значение 1>
```

Если же предложение ROWS задано в виде:

```
ROWS <значение 1> TO <значение 2>
```

то ему соответствует следующая конструкция из ключевых слов FIRST и SKIP:

```
FIRST (<значение 2> - <значение 1> + 1)
SKIP (<значение 1> - 1)
```

Не существует предложения ROWS, соответствующего тому случаю, когда заданы и ключевое слово FIRST, и ключевое слово SKIP.

## Предложение 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:

```
[OFFSET <значение1> {ROW | ROWS}]
[FETCH {FIRST | NEXT} [<значение2>] {ROW | ROWS} ONLY]
```

где аргументами могут быть числовые литералы, SQL параметры (?) или PSQL параметры (:param).

Предложение FETCH, OFFSET можно использовать, только если задано и предложение ORDER BY. Предложения OFFSET и/или FETCH не могут быть объединены с предложениями ROWS или FIRST/SKIP в одном выражении запроса.

## Предложение FOR UPDATE

Предложение FOR UPDATE не делает то, что от него ожидается. В настоящее время единственный эффект от его использования заключается лишь в отключении упреждающей выборки в буфер.

Предложение OF не делает ничего вообще.

## Предложение 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.

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 8.5 — 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.<br>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. |
| `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.<br>Для этого режима TPB никогда не возникает конфликта обновления |

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.

### Предложение 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.

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.

## 8.2 INSERT

Добавление новых строк в обычную таблицу или в таблицу, лежащую в основе представления, осуществляется с помощью оператора INSERT. Его синтаксис представлен в листинге 8.15.

**Листинг 8.15. Синтаксис оператора добавления данных INSERT**

```
INSERT INTO {<имя таблицы> | <имя представления>} {
    [(<список столбцов>)] {VALUES (<значение> [,<значение> ...])|<поиск многих>}
  | DEFAULT VALUES }
[RETURNING <имя столбца> [[AS] <алиас>] [, <имя столбца> [[AS] <алиас>]...]
   [INTO [:]<имя переменной> [, [:]<имя переменной> ...]] ];

<список столбцов> ::= <имя столбца> [, <имя столбца> ...]
```

Оператор позволяет добавлять строки в обычную таблицу или в таблицу изменяемого представления — см. главу 9 «Работа с представлениями».

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, или строку, которая будет во всех столбцах содержать значения по умолчанию (предложение DEFAULT VALUES).

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.

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.

Для оператора INSERT существует два варианта — либо помещаемые в строку данные содержатся в самом операторе (используется ключевое слово VALUES; при этом значения отдельных столбцов могут быть получены из базы данных при выполнении оператора SELECT), либо все данные добавляемой строки выбираются при помощи сколь угодно сложного оператора SELECT из какой-либо таблицы или группы таблиц, представления или хранимой процедуры выбора. Для использова-

ния 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».

## Использование ключевого слова 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. Это соответствует первому варианту оператора добавления данных (см. листинг 8.16):

**Листинг 8.16. Первый вариант оператора добавления данных INSERT**

```
INSERT INTO {<имя таблицы> | <имя представления>}
   [(<список столбцов>)] VALUES (<значение> [,<значение> ...])
[RETURNING <имя столбца> [[AS] <алиас>] [, <имя столбца> [[AS] <алиас>]...]
   [INTO [:]<имя переменной> [, [:]<имя переменной> ...]] ];
```

**Листинг 8.17. Синтаксис значения в операторе INSERT**

```
<значение> ::= {
    <литерал>
  | <выражение>
  | <встроенная функция>
  | <UDF> [(<параметр> [, <параметр> ...])]
  | <обращение к хранимой процедуре> [(<параметр> [, <параметр>]...)]
  | NEXT VALUE FOR <имя генератора>
  | (<выбор одного>) }
```

Чаще всего значениями являются литералы, то есть самоопределенные константы, предварительно определенные литералы, контекстные переменные.

Подробные описания характеристик и порядка использования предварительно определенных литералов и контекстных переменных см. в главе 2 «Типы данных Ред База Данных», а также в приложении Ж «Контекстные переменные».

Значением может быть и правильное выражение любой сложности. Оно, в частности, может содержать и оператор SELECT, выбирающий одно значение или пустое значение NULL из таблицы, представления или хранимой процедуры выбора. Этот оператор во всех конструкциях должен быть заключен в круглые скобки.

В операторе допустимо использование встроенных и определенных пользователем функций (UDF — подробности см. в приложении Г), которые возвращают в точности одно значение. Функциям могут передаваться параметры.

Для формирования значений искусственных первичных ключей может использоваться встроенная функция GEN_ID или конструкция NEXT VALUE FOR — см. далее.

В SQL Ред База Данных существует два типа встроенных функций — обычные встроенные функции и агрегатные функции в операторе SELECT.

Обычная встроенная функция — это функция, получающая один или более параметров, которая не связана с оператором выборки данных SELECT. Функция возвращает ровно одно значение. Параметры передаются таким функциям на основании принятого для каждой функции синтаксиса. Описание встроенных функций см. в приложении Е «Функции».

В агрегатной функции оператор SELECT выбирает из указанной таблицы на основании за-

данного условия некоторое количество значений. Агрегатная функция внутри оператора SELECT выполняет соответствующие расчеты и возвращает одно число.

- Функция COUNT подсчитывает количество указанных объектов;
- Функция SUM возвращает сумму указанных значений полученных строк;
- Функция AVG возвращает среднее арифметическое значение указанных значений полученных строк;
- Функции MAX и MIN задают, соответственно, поиск максимального и минимального значения среди всех значений полученных строк. Функции могут работать с любыми типами данных кроме BLOB;
- Функция LIST объединяет в один объект типа BLOB все данные, полученные из указанного выражения;
- Функции CORR, COVAR_POP, COVAR_SAMP, STDDEV_POP, STDDEV_SAMP, VAR_POP, VAR_SAMP используются для получения статистических показателей;
- Функции линейной регрессии REGR_AVGX, REGR_AVGY, REGR_COUNT, REGR_INTERCEPT, REGR_R2, REGR_SLOPE, REGR_SXX, REGR_SXY и REGR_SYY 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.

Подробные описания агрегатных функций см. в главе 4 «Работа с доменами» и в приложении E «Функции».

Конструкция NEXT VALUE FOR используется вместо функции GEN_ID. Использование этой функции является более предпочтительным в последних версиях сервера базы данных.

Здесь также может быть использован оператор SELECT, возвращающий единственное значение на основании условий поиска в таблице, в группе таблиц, представления или при вызове хранимой процедуры выбора. Этот оператор должен быть заключен в круглые скобки. При использовании оператора SELECT пользователь должен иметь соответствующие полномочия к данным используемых таблиц.

**Пример.** Если нужно поместить новую запись в таблицу PEOPLE, содержащую сведения о людях, где присутствует искусственный первичный ключ PEOPLE_ID, то для получения значения первичного ключа можно применить встроенную функцию GEN_ID, которая, используя созданный перед этим генератор, возвращает уникальное значение искусственного первичного ключа:

```
INSERT INTO PEOPLE (PEOPLE_ID, ..., LAST_NAME)
   VALUES (GEN_ID(GEN_PEOPLE, 1), ..., 'Рустамов');
```

Синтаксис встроенной функции GEN_ID следующий:

```
GEN_ID(<имя генератора>, <приращение>)
```

В качестве приращения обычно выбирается единица.

В настоящей версии Ред База Данных рекомендуется вместо этой встроенной функции использовать конструкцию SQL NEXT VALUE FOR. В этой конструкции значение генератора всегда увеличивается точно на единицу.

**Пример.** Предыдущий оператор может быть изменен при использовании конструкции NEXT VALUE FOR следующим образом:

```
INSERT INTO PEOPLE (PEOPLE_ID, ..., LAST_NAME)
   VALUES (NEXT VALUE FOR GEN_PEOPLE, ..., 'Рустамов');
```

Пример использования оператора SELECT для получения одного значения столбца добавляемой строки таблицы. При помещении строки в таблицу регионов REGION нужно коду страны CODCOUNTRY присвоить соответствующее значение. Для этого используется оператор SELECT, обращающийся к таблице стран COUNTRY.

```
INSERT INTO REGION (CODCOUNTRY, CODREGION, ...)
   VALUES ( (SELECT CODCOUNTRY
             FROM COUNTRY
             WHERE NAME = 'РОССИЯ'), '64', ...);
```

Оператор SELECT заключен в дополнительные круглые скобки. Этот оператор выбирает код страны, используя краткое название страны. Такой оператор удобен, если вы не помните все коды стран, с которыми работаете, или коды стран в вашей базе данных часто меняются. В этом случае такой оператор будет инвариантным по отношению к изменяемым кодам стран.

## Использование поиска многих

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. Синтаксис второго, пакетного, варианта оператора добавления данных показан в листинге 8.18.

**Листинг 8.18. Второй вариант оператора добавления данных INSERT**

```
INSERT INTO {<имя таблицы> | <имя представления>}
   [(<список столбцов>)] <поиск многих>
[RETURNING <имя столбца> [[AS] <алиас>] [, <имя столбца> [[AS] <алиас>]...]
   [INTO [:]<имя переменной> [, [:]<имя переменной> ...]] ];
```

В конструкции «поиск многих» используется оператор SELECT, который выбирает необходимые данные из таблицы базы данных, представления или при обращении к хранимой процедуре выбора.

Для того чтобы иметь возможность выбирать данные при использовании подобного оператора SELECT, пользователь должен иметь соответствующие полномочия к таблицам.

В этом варианте 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 — см. раздел «SELECT». Список выбора может содержать не только имена столбцов, но и литералы.

Предложение RETURNING может быть использовано только в том случае, если оператор SELECT выбирает только одну запись. Иначе будет выдано сообщение об ошибке.

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. следующий раздел, главу 5 «Работа с таблицами», главу 11 «Хранимые процедуры, хранимые функции, триггеры и пакеты».

### Использование 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.

### Предложение 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 — см. главу 11 «Хранимые процедуры, хранимые функции, триггеры и пакеты».

## 8.3 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 8.19.

**Листинг 8.19. Синтаксис оператора изменения данных UPDATE**

```
UPDATE {<имя таблицы> | <имя представления>} [[AS] <псевдоним>]
SET <имя столбца> = <значение> [, <имя столбца> = <значение> ...]
[WHERE { <условие поиска> | CURRENT OF <имя курсора>}]
[PLAN <план>]
```

```
[ORDER BY <упорядочиваемый элемент> [, <упорядочиваемый элемент> ... ]]
[ROWS <значение 1> [TO <значение 2>]]
[RETURNING <имя столбца> [[AS] <алиас>] [, <имя столбца> [[AS] <алиас>]...]
   [INTO [:]<имя переменной> [, [:]<имя переменной> ...]] ];
```

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.

## Предложение SET

Предложение SET задает список выполняемых изменений: указывается имя изменяемого столбца и после знака равенства новое значение столбца. Значением, как и в случае оператора INSERT, может быть сколь угодно сложное выражение. Значение определяется следующим синтаксисом (см. листинг 8.20).

**Листинг 8.20. Синтаксис значения в операторе изменения данных UPDATE**

```
<значение> ::= {
    <литерал>
  | <выражение>
  | <встроенная функция>
  | <UDF> [(<параметр> [, <параметр> ...])]
  | NEXT VALUE FOR <имя генератора>
  | (<выбор одного>) }
```

Описание возможных вариантов значения см. в разделе «INSERT».

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 |

Здесь стоит обратить внимание на то, что для обновления столбца B используются старые значения

столбца A. Так было не всегда. До версии 2.5, столбцы сразу получали новые значения. Это являлось нестандартным поведением, и поэтому было изменено в версии 2.5. Однако, для восстановления совместимости со старыми версиями в firebird.conf существует параметр OldSetClauseSemantics, который, будучи установленным в 1, восстанавливает старое поведение. Этот параметр в будущем будет удален.

## Предложение 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.

Подробнее об условиях поиска в предложении WHERE см. в разделе «SELECT».

## Предложение PLAN

В операторе изменения данных может быть использовано ключевое слово PLAN, задающее план выборки данных. Подробнее о планах выборки см. в разделе «SELECT». 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 наиболее подходящий план для поиска записей. При достаточном опыте работы с базой данных вы можете задавать свой собственный план, позволяющий ускорить работу системы.

## Предложение ORDER BY и ROWS

Предложение ORDER BY имеет смысл использовать только в том случае, если далее будет задано предложение ROWS. Предложение ORDER BY используется для упорядочения результатов выборки. В нем указывается список столбцов, по которым происходит упорядочение, направление сортировки (по возрастанию или по убыванию) и порядок сортировки для строковых столбцов, если требуемый порядок сортировки отличается от принятого для всей базы данных или для данного столбца. После упорядочения выполняются изменения для указанных строк. Формально синтаксис для ORDER BY выглядит следующим образом (листинг 8.21):

**Листинг 8.21. Синтаксис предложения упорядочения данных ORDER BY**

```
ORDER BY <упорядочиваемый элемент> [, <упорядочиваемый элемент> ... ]
[ROWS <значение 1> [TO <значение 2>]]

<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 приложении В «Наборы символов и порядок сортировки».

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.

Предложение ROWS задает диапазон строк, к которым будет применена операция изменения данных. Предложение ROWS можно использовать, только если задано и предложение ORDER BY.

```
ROWS <значение 1> [TO <значение 2>]
```

Значением здесь может быть число или выражение, возвращающее числовое значение. Число может быть и дробным, в этом случае десятичные знаки просто отбрасываются без округления числа. Если используется выражение, то оно должно быть заключено в круглые скобки. Если в выражении присутствует оператор SELECT, то он дополнительно должен быть заключен в круглые скобки.

Значение 1 задает количество включаемых в операцию обновления строк, упорядоченных в предложении ORDER BY, если не задан вариант TO. Это первые строки в упорядоченном списке.

Если указан и вариант TO, то значение 1 задает начальный номер строки в упорядоченном списке строк. Значение 2 в этом случае задает конечный номер выбираемой строки.

Для возможности использования предложения ORDER BY в операторе UPDATE пользователь должен также иметь полномочия выборки данных из соответствующей таблицы — GRANT SELECT.

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.

## Предложение RETURNING

Необязательное предложение RETURNING указывает, что оператор возвращает значения заданных столбцов изменяемой 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. Ключевое слово INTO позволяет сохранить эти возвращенные значения во внутренних переменных триггера или хранимой процедуры.

## Примеры

Последний пример из предыдущего раздела, где регионы из одной страны переносились в другую страну, проще (и естественнее) переписать с использованием одного единственного оператора UPDATE:

```
UPDATE REGION
SET CODCOUNTRY = 'USA'
WHERE (CODCOUNTRY = 'RUS' AND CODREGION = 'TX');
```

Первичным ключом таблицы регионов является составной ключ, содержащий два столбца — код страны (CODCOUNTRY) и код региона (CODREGION). Данный оператор UPDATE просто изменяет значение первичного ключа таблицы.

Если таблица регионов содержит подчиненные таблицы, например, таблицу районов, где описан внешний ключ, ссылающийся на первичный ключ таблицы регионов, и при описании внешнего ключа было задано предложение ON UPDATE CASCADE, то в результате выполнения этой операции значения внешних ключей подчиненной таблицы будут автоматически приведены в соответствие с изменившимся значением первичного ключа таблицы регионов. Эти изменения выполнит автоматически созданный системой триггер.

Пусть нужно изменить код страны у нескольких записей регионов. Можно выполнить следующий оператор:

```
UPDATE REGION
SET CODCOUNTRY = 'RUS'
ORDER BY NAMEREG NULLS LAST
ROWS 2 TO 3;
```

Здесь строки таблицы перед изменением упорядочиваются по именам регионов. Изменение кода страны происходит только у второй и третьей строк полученного из исходной таблицы упорядоченного списка.

Пример использования изменения в случае применения предложения `WHERE`, предложения `ORDER BY` и соответствующего ему предложения `ROWS`.

```
UPDATE REGION
SET CODCOUNTRY = 'RUS'
WHERE CODCOUNTRY = 'ENG'
ORDER BY NAMEREG
ROWS 4;
```

Здесь для изменения отбираются только те строки, у которых код страны имеет значение `'ENG'`, эти строки упорядочиваются по названию региона, затем изменения вносятся в первые четыре строки.

Для этого оператора сервером базы данных будет создан план:

```
PLAN SORT (FIRM NATURAL)
```

Подробно планы выборки рассматриваются в разделе «SELECT».

# 8.4 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 8.22.

**Листинг 8.22. Синтаксис оператора изменения или добавления данных `UPDATE OR INSERT`**

```
UPDATE OR INSERT INTO
   {<имя таблицы> | <имя представления>} [(<имя столбца> [, <имя столбца> ...] )]
VALUES (<значение> [, <значение> ...] )
[MATCHING (<имя столбца> [, <имя столбца>] ...)
[RETURNING <имя столбца> [[AS] <алиас>] [, <имя столбца> [[AS] <алиас>]...]
   [INTO [:]<имя переменной> [, [:]<имя переменной> ...]] ];
```

Для выполнения оператора `UPDATE OR INSERT` пользователь должен иметь полномочия и `UPDATE`, и `INSERT` к таблице (представлению).

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.

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.

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. 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 — см. главу 11 «Хранимые процедуры, хранимые функции, триггеры и пакеты».

# 8.5 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 8.23.

**Листинг 8.23. 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 <значение 1> [TO <значение 2>]]
[RETURNING <имя столбца> [[AS] <алиас>] [, <имя столбца> [[AS] <алиас>]...]
   [INTO [:]<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.

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 «SELECT».

Может быть использовано ключевое слово PLAN, задающее план выборки данных для удаления. Подробнее о плане выборки см. в разделе «SELECT». 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.

Предложение ORDER BY следует использовать, если далее будет задано предложение ROWS. Предложение ORDER BY используется для упорядочения результатов выборки. 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 8.3. Изменение данных. Оператор UPDATE этой главы.

Предложение RETURNING позволяет вернуть вызвавшей программе значения указанных столбцов удаленной строки. Если происходит удаление более чем одной строки, то выдается сообщение об ошибке. Ключевое слово INTO позволяет сохранить возвращенные значения во внутренних переменных триггера или хранимой процедуры. Вариант INTO может использоваться только в PSQL — см. главу 11 «Хранимые процедуры, хранимые функции, триггеры и пакеты».

## Примеры

Если надо удалить один конкретный регион из таблицы регионов, то следует выполнить оператор:

```
DELETE FROM REGION
WHERE (CODCOUNTRY = 'RUS' AND CODREGION = 'TX');
```

Если удаляемый регион содержит сведения о районах в другой таблице и внешний ключ в этой таблице содержит предложение ON DELETE CASCADE, то будут удалены все соответствующие строки районов. Пользователь, выполняющий это удаление должен иметь полномочия и для удаления строк таблицы районов.

Для удаления всех регионов одной страны используется оператор:

```
DELETE FROM REGION
WHERE CODCOUNTRY = 'RUS';
```

Пользователь также должен иметь полномочия по удалению и районов стран.

Для удаления всех регионов всех стран нужно использовать оператор удаления без предложения WHERE:

```
DELETE FROM REGION;
```

Для удаления группы заданных строк из упорядоченного списка, полученного из таблицы регионов, следует выполнить оператор:

```
DELETE FROM REGION
WHERE CODCOUNTRY = 'RUS'
ORDER BY NAMEREG DESCENDING NULLS FIRST
ROWS 2 TO 3;
```

Здесь выбираются все регионы, содержащие в столбце кода страны значение 'RUS', затем записи упорядочиваются в порядке убывания названий регионов, после этого удаляется вторая и третья строка полученного списка.

Следующий оператор удаляет первые четыре строки из упорядоченного списка регионов, относящихся к стране с кодом 'ENG':

```
DELETE FROM REGION
WHERE CODCOUNTRY = 'ENG'
ORDER BY NAMEREG
ROWS 4;
```

Если список будет содержать менее четырех строк, то будут удалены существующие строки без выдачи диагностических сообщений.

В этом случае сервер базы данных составит следующий план:

```
PLAN SORT (REGION INDEX (FK_REGION))
```

Здесь выборка строк регионов будет осуществляться с использованием автоматически созданного индекса внешнего ключа таблицы, который ссылается на первичный ключ таблицы стран.

Подробно планы выборки рассматриваются в разделе «SELECT».

# 8.6 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 8.24.

**Листинг 8.24. Синтаксис оператора EXECUTE BLOCK**

```
EXECUTE BLOCK
   [(<список входных параметров>)]
   [RETURNS (<список выходных параметров>)]
AS
   [<объявление локальной переменной> [<объявление локальной переменной> ...] ]
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 локальной переменной> ::=
   DECLARE [VARIABLE] {
     <имя локальной переменной> <тип>
     [NOT NULL]
```

```
    [COLLATE <порядок сортировки>]
    [{ = | DEFAULT } <значение по умолчанию>]
  | <имя курсора> CURSOR FOR (<оператор SELECT>) }
```

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.

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 переменной) 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.

**Пример 8.31**

```
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 11 «Хранимые процедуры и триггеры». Там же приведен пример хранимой процедуры, выполняющей вычисление факториала заданного числа.

Глава 9

# Работа с представлениями

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.

## 9.1 Создание представлений

Для создания представления используется оператор CREATE VIEW. Синтаксис оператора показан в листинге 9.1.

**Листинг 9.1. Синтаксис оператора создания представления CREATE VIEW**

```
CREATE VIEW <имя представления>
   [( <имя столбца> [AS <псевдоним>] [, <имя столбца> [AS <псевдоним>] ...] )]
AS <оператор SELECT>
[WITH CHECK OPTION];
```

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. Подробнее о привилегиях см. в документе «Руководство администратора».

Имя представления должно быть уникальным среди имен всех представлений, таблиц и хранимых процедур базы данных.

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.

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 разделе 8.1 «SELECT».

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 9.6 Примеры представлений 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, допускают пустые значения NULL;
- 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.);
- 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.

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.

## 9.2 Изменение представлений

Для изменения существующего представления используется оператор ALTER VIEW. Синтаксис оператора показан в листинге 9.2.

**Листинг 9.2. Синтаксис оператора создания представления ALTER VIEW**

```
ALTER VIEW <имя представления>
   [( <имя столбца> [AS <псевдоним>] [, <имя столбца> [AS <псевдоним>] ...] )]
AS <оператор SELECT>
[WITH CHECK OPTION];
```

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.

## 9.3 Создание или изменение представлений

Для создание нового или изменение существующего представления используется оператор CREATE OR ALTER VIEW. Синтаксис оператора показан в листинге 9.3.

**Листинг 9.3. Синтаксис оператора создания представления CREATE OR ALTER VIEW**

```
CREATE OR ALTER VIEW <имя представления>
   [( <имя столбца> [AS <псевдоним>] [, <имя столбца> [AS <псевдоним>] ...] )]
AS <оператор SELECT>
[WITH CHECK OPTION];
```

Оператор CREATE OR ALTER VIEW создаёт представление, если оно не существует. В противном случае он изменит представление с сохранением существующих зависимостей.

## 9.4 Удаление представлений

Для удаления существующего в базе данных представления используется оператор DROP VIEW. Его синтаксис представлен в листинге 9.4.

**Листинг 9.4. 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.

## 9.5 Пересоздание представлений

Оператор RECREATE VIEW позволяет внести изменения в существующее представление.

**Листинг 9.5. Синтаксис оператора пересоздания представления RECREATE VIEW**

```
RECREATE VIEW <имя представления>
   [( <имя столбца> [AS <псевдоним>] [, <имя столбца> [AS <псевдоним>] ...] )]
AS <оператор SELECT>
[WITH CHECK OPTION];
```

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.

Представление может заново создать его создатель и любой пользователь с ролью RDB$ADMIN.

## 9.6 Примеры представлений

Любой пример оператора SELECT из главы 8 можно записать в виде представления.

**Пример 1.** В примере 8.1 главы 8 был приведен пример использования производной таблицы. Можно создать соответствующее представление USER_TABLES. Это представление отображает все пользовательские таблицы базы данных.

**Пример 9.1**

```
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 8.13 в главе 8. Соответствующий оператор SELECT представления также содержит два левых внешних соединения с той же таблицей.

**Пример 9.2**

```
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 9.7 «Преобразование неизменяемых представлений в изменяемые при помощи триггеров» 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`:

**Пример 9.3**

```
SELECT
   C1 AS "Фамилия, имя, отчество",
   C2 AS "Мать",
   C3 AS "Отец"
FROM SELECT_PEOPLE
ORDER BY C1;
```

Здесь будет получен в точности такой же результат, что и при выполнении оператора `SELECT`, заданного в листинге 8.13 в главе 8.

**Пример 3.** Следующий несколько надуманный пример (Пример 9.4) 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.

**Пример 9.4**

```
CREATE VIEW V_TEST (CODCOUNTRY, NAME, FULLNAME, CAPITAL, DESCR)
AS
   SELECT *
   FROM COUNTRY
   WHERE CODCOUNTRY CONTAINING '5'
WITH CHECK OPTION;
```

Здесь выбираются все страны из таблицы стран, у которых в коде страны присутствует символ «5» — 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 «5».

**Пример 4.** В следующем простом представлении выбираются все регионы всех стран, хранящиеся в базе данных.

**Пример 9.5**

```
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:

**Пример 9.6**

```
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 примере 9.7. Здесь код страны 'РОССИЯ' получается при помощи внутреннего оператора SELECT, который получает код на основании заданного краткого названия страны.

**Пример 9.7**

```
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:

**Пример 9.8**

```
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:

**Пример 9.9**

```
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 назва-

ние 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 (см. Пример 9.10). 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.

**Пример 9.10**

```
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.

**Пример 9.11**

```
DELETE FROM VIEW_RUSSIA2
WHERE CODREGION = '00' AND
   CODCOUNTRY = (SELECT CODCOUNTRY
                 FROM COUNTRY
                 WHERE NAME = 'РОССИЯ');
```

## 9.7 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 (см. Пример 9.12). При этом представление включает в себя все столбцы базовых таблиц, которые не могут принимать пустые значения. Следовательно, существует принципиальная возможность сделать это представление изменяемым.

**Пример 9.12**

```
CREATE VIEW V_REGION_COUNTRY
   (CODCOUNTRY, CODREGION, NAMEREG, CENTER, DESCR, NAME, FULLNAME)
AS
   SELECT
     REGION.CODCOUNTRY,
     CODREGION,
     NAMEREG,
     CENTER,
     REGION.DESCR,
     COUNTRY.NAME,
     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 Примере 9.13. 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.

**Пример 9.13**

```
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.

**Пример 9.14**

```
SET TERM ^;
CREATE TRIGGER TBU_REGION_COUNTRY
   FOR V_REGION_COUNTRY
   BEFORE UPDATE
AS
BEGIN
   UPDATE REGION
   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.

**Пример 9.15**

```
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 = 'ENG';
```

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:

**Пример 9.16**

```
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 11 «Хранимые процедуры, хранимые функции, триггеры и пакеты». 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 происходи 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 11 «Хранимые процедуры, хранимые функции, триггеры и пакеты».

## 9.8 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 примере 9.17, 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.

**Пример 9.17**

```
CREATE VIEW CHECK_CONSTRAINTS (CONSTRAINT_NAME, CHECK_CLAUSE)
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:

**Пример 9.18**

```
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.

**Пример 9.19**

```
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:

**Пример 9.20**

```
SELECT
   TABLE_NAME AS "Таблица",
   COLUMN_NAME AS "Столбец",
   CONSTRAINT_NAME AS "Имя ограничения"
FROM CONSTRAINTS_COLUMN_USAGE;
```

**Пример 3.** В примере 9.21 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.

**Пример 9.21**

```
CREATE VIEW REFERENTIAL_CONSTRAINTS
   (CONSTRAINT_NAME, UNIQUE_CONSTRAINT_NAME, MATCH_OPTION,
    UPDATE_RULE, DELETE_RULE)
AS
   SELECT
     RDB$CONSTRAINT_NAME,
     RDB$CONST_NAME_UQ,
     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:

**Пример 9.22**

```
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 (см. Пример 9.23) 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.

**Пример 9.23**

```
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:

**Пример 9.24**

```
SELECT
   CONSTRAINT_NAME AS "Имя ограничения",
   TABLE_NAME AS "Таблица",
   CONSTRAINT_TYPE AS "Вид ограничения",
   IS_DEFERRABLE,
   INITIALLY_DEFERRED
FROM TABLE_CONSTRAINTS;
```

## 9.9 Примечание представления

Для представления можно создать примечание, используя следующий синтаксис оператора COMMENT:

**Листинг 9.6. 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 10

# Транзакции

Транзакции являются особым механизмом базы данных. Клиентские процессы напрямую не взаимодействуют с данными базы данных, хранящимися в страницах файла (файлов) базы данных. Вместо этого все процессы взаимодействия с базой данных осуществляются при использовании специальных средств связи клиентов с базой данных. Все действия с объектами базы данных (с метаданными) и с данными, хранимыми в базе данных, выполняются под управлением какой-либо транзакции или, как еще говорят, в контексте некоторой транзакции. Все изменения, удаления и добавления данных, выполняемые в рамках данной транзакции, изолированы от действий, выполняемых в других параллельных процессах в контекстах других транзакций. Текущая транзакция может рассматриваться как единственный изолированный от других параллельных процессов отдельный процесс, выполняющий изменение данных в базе данных. Все изменения в базе данных, выполненные в контексте одной транзакции, можно либо подтвердить (для этого используется оператор `COMMIT`), тогда они становятся видимыми для других параллельных процессов, либо отменить (оператор `ROLLBACK`). Транзакция переводит одно непротиворечивое состояние базы данных в другое непротиворечивое состояние (если база данных правильно спроектирована, заданы соответствующие ограничения для доменов, столбцов таблиц и для таблиц в целом, созданы необходимые триггеры).

Еще одно определение транзакции — это завершенное общение клиента с сервером базы данных.

Подводя итог, можно сказать, что транзакция — это механизм, позволяющий объединить группу действий, выполняемых с данными или метаданными базы данных, в один логический блок. Все действия, выполненные в одном блоке, можно или подтвердить или отменить. Действия в рамках одной транзакции переводят базу данных из одного непротиворечивого состояния в другое непротиворечивое состояние.

Запуск транзакции осуществляется при помощи оператора `SET TRANSACTION`.

В процессе выполнения одной транзакции существует возможность создавать именованные контрольные точки (оператор `SAVEPOINT`). В дальнейшей работе с базой данных можно выполнять откат на любую из созданных контрольных точек (`ROLLBACK TO SAVEPOINT`), а не только на самое начало транзакции. Любую из созданных контрольных точек можно удалить (`RELEASE SAVEPOINT`). Такое средство использования контрольных точек называется в литературе также вложенными транзакциями (`nested transactions`).

Классическими проблемами при наличии нескольких клиентов, одновременно работающих с одной базой данных на сервере, являются следующие.

*Потеря изменений* (`lost updates`). Возникает, когда один клиент изменяет данные в одной строке таблицы и подтверждает эти изменения, а другой клиент вскоре после этого вносит иные изменения в ту же самую строку. Сохранятся последние изменения. Первый клиент без переоткрытия набора данных (а в некоторых случаях и без перезапуска транзакции) не увидит последние изменения, выполненные вторым клиентом. Вряд ли это следует рассматривать как проблему. Всегда в базе данных будут сохраняться последние изменения данных, если конкурирующие процессы имеют право на изменение данных. Для получения актуального состояния базы данных следует выполнять переоткрытие набора данных и иногда перезапуск транзакции (необходимость перезапуска зависит от уровня изоляции транзакции). Если же действительно нужен монопольный доступ ко всем или к некоторым таблицам в контексте одной транзакции, чтобы другие процессы не могли изменять (а иногда и читать) важные для решения задач предметной области данные, то следует либо использовать предложение `WITH LOCK` в операторе `SELECT` при выборе данных из одной конкретной таблицы, либо применить уровень изоляции для транзакции `SNAPSHOT TABLE STABILITY`, если требуется запретить любые изменения всех таблиц, используемых в данной транзакции, либо при старте транзакции использовать предложение резервирования таблиц `RESERVING`.

*Невоспроизводимые чтения* (`non-reproducible reads`). Один клиент читает старые версии

уже измененных другими параллельными процессами записей, не видя актуального состояния базы данных, в том числе и новых строк. Такая ситуация возникает при использовании уровня изоляции транзакции `SNAPSHOT` (мгновенный снимок базы данных). В случае использования уровня изоляции `READ COMMITTED` (подтвержденное чтение, то есть чтение подтвержденных изменений базы данных другими транзакциями) такое происходит много реже — только в том случае, если набор данных не открывается повторно при сохранении контекста такой транзакции.

*Фантомные строки* (`phantom rows`). Это строки, удаленные в других параллельных процессах, но которые все еще видны в текущей транзакции. Опять же, это естественное явление для уровня изоляции `SNAPSHOT`. Для `READ COMMITTED` такие строки появляются гораздо реже — чтобы удаленные строки перестали быть видимыми в этой транзакции, нужно просто переоткрыть набор данных в контексте такой транзакции.

*Перекрывающиеся транзакции* (`interleaved transactions`), или побочные эффекты изменений (`update side effects`). В этом случае изменение данных одним клиентом приводят к нарушениям базы данных, которые видимы другим клиентам. Решением данной проблемы, как и многих других, является помещение всех операций, которые по отдельности могут привести к несогласованности данных в базе данных, в контекст одной транзакции. Классическим примером является перевод денег с одного счета на другой в пределах одного финансового учреждения. Если в одной транзакции выполняется снятие денег с одного счета, а в другой транзакции того же клиентского процесса — зачисление этой суммы на иной счет, то после подтверждения первой транзакции база данных будет находиться в несогласованном (с содержательной точки зрения предметной области) состоянии. Любой параллельный процесс получит базу данных в таком неверном состоянии. При этом декларативная, то есть формальная целостность базы данных будет сохранена. В подобном случае следует операции снятия денег с одного счета и зачисления их на другой счет выполнять в контексте одной неделимой транзакции, а не двух или более.

Есть еще теоретическая проблема *грязного чтения* (`dirty read`), когда транзакция может видеть неподтвержденные изменения строк. В СУБД Ред База Данных такой проблемы не существует. Неподтвержденные изменения нельзя увидеть ни в какой транзакции с любым уровнем изоляции.[1]

В зависимости от требований обработки данных в конкретной предметной области СУБД Ред База Данных позволяет выбрать такую конфигурацию характеристик используемых транзакций, которая наилучшим образом решит большинство возникающих проблем.

При работе с базой данных клиентские программы могут использовать как длинную, так и короткую транзакцию. Длинная транзакция является активной в течение довольно большого промежутка времени. В ее контексте может выполняться большое количество операций с базой данных. Коротка транзакция от момента ее старта и до момента ее подтверждения или отката является активной доли секунды. Как правило, в контексте такой транзакции выполняется весьма ограниченное количество изменений в базе данных.

Длинные транзакции обычно используются в случае чтения данных базы данных. Короткие транзакции являются наиболее подходящими в случае внесения изменений в таблицы базы данных.

## 10.1 Старт транзакции

Для запуска (старта) транзакции используется оператор `SET TRANSACTION`. Его синтаксис представлен в листинге 10.1. Запуск транзакций на выполнение осуществляется только клиентскими приложениями, но не сервером. Каждое клиентское приложение может запускать произвольное количество одновременно выполняющихся транзакций.[2] Фактически есть ограничение на общее количество выполняемых транзакций во всех клиентских приложениях, работающих с одной конкретной базой данных с момента последнего восстановления базы данных с резервной копии или с момента первоначального создания базы данных. Это количество равняется числу $2^{48} - 1$, то есть $\approx 2,8 \times 10^{14}$.

[1] Приведенный список проблем, возникающих при использовании баз данных в архитектуре клиент-сервер, не всегда соответствует по интерпретации некоторым существующим литературным источникам.

[2] В динамическом SQL (DSQL), который используется в isql и в соответствующих программах графического интерфейса, в каждый момент времени может быть запущена только одна транзакция. В DSQL также не используются именованные транзакции.

**Листинг 10.1. Синтаксис оператора запуска транзакции SET TRANSACTION**

```
SET TRANSACTION
[NAME <имя транзакции>]
[READ WRITE | READ ONLY]
[WAIT [LOCK TIMEOUT <кол-во секунд>] | NO WAIT]
[[ISOLATION LEVEL]
   { SNAPSHOT [TABLE STABILITY] | READ COMMITTED [[NO] RECORD_VERSION] } ]
[NO AUTO UNDO]
[IGNORE LIMBO]
[RESERVING <предложение резервирования> | USING <хендл базы данных>];

<предложение резервирования> ::=
   <имя таблицы> [, <имя таблицы> ...]
   [FOR [SHARED | PROTECTED] {READ | WRITE}]
   [, <предложение резервирования>] ...
```

Все предложения в операторе `SET TRANSACTION` являются необязательными. Если в операторе запуска транзакции на выполнение не задано никакого предложения, то предполагается старт транзакции со значениями всех характеристик по умолчанию (режим доступа, режим разрешения блокировок и уровень изоляции):

```
SET TRANSACTION
READ WRITE
WAIT
ISOLATION LEVEL SNAPSHOT;
```

Транзакция с такими же характеристиками по умолчанию автоматически запускается системой, если пользователь, выполняя обращение к данным базы данных, не запустил никакой транзакции.

При старте со стороны клиента любой транзакции (заданной явно или по умолчанию) сервер передает клиенту дескриптор транзакции (целое число). Значение этого дескриптора средствами SQL можно получить, используя контекстную переменную `CURRENT_TRANSACTION`. О контекстных переменных см. в главе 2 «Типы данных Ред База Данных».

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 (`ISOLATION LEVEL`) и средства резервирования или освобождения таблиц (предложение `RESERVING`).

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 спецификацию имён транзакций.

## Средства резервирования

Предложение `RESERVING` в операторе запуска транзакции резервирует указанные в списке таблицы, то есть запрещает другим транзакциям вносить в эти таблицы изменения или (при определенных установках характеристик предложения резервирования) даже читать данные из этих таблиц в то время как выполняется данная транзакция. Либо, наоборот, в этом предложении можно указать список таблиц, в которые параллельные процессы могут вносить изменения даже если запускается транзакция с уровнем изоляции `SNAPSHOT TABLE STABILITY`. Синтаксис предложения резервирования представлен в листинге 10.2.

**Листинг 10.2. Синтаксис предложения резервирования**

```
RESERVING <предложение резервирования>

<предложение резервирования> ::=
   <имя таблицы> [, <имя таблицы> ...]
   [FOR [SHARED | PROTECTED] {READ | WRITE}]
   [, <предложение резервирования>] ...
```

Если опущено ключевые слова SHARED и PROTECTED, то предполагается SHARED. Если опущено все предложение FOR, то предполагается FOR SHARED READ. Варианты осуществления резервирования таблиц по их названиям не являются очевидными. Воздействие различных способов резервирования таблиц на другие параллельно выполняемые транзакции рассматриваются более подробно далее при описании различных уровней изоляции транзакций.

Указанное резервирование осуществляется сразу же в момент старта транзакции. В операторе SELECT, выбирающем данные из таблиц, существует возможность задания необязательного предложения WITH LOCK. (см. раздел 8.1 «SELECT»). Использование этого предложения «закрывает», блокирует, таблицу, из которой осуществляется выборка данных, от любых изменений другими одновременно выполняющимися процессами. При этом в отличие от предложения резервирования RESERVING в операторе старта транзакции в предложении WITH LOCK оператора SELECT такая блокировка начинает действовать только с момента выполнения этого запроса к базе данных. Завершение такой блокировки осуществляется также по завершении транзакции, в контексте которой выполнялся этот оператор SELECT.

В одном предложении резервирования можно указать произвольное количество резервируемых таблиц используемой базы данных.

Для уровней изоляции SNAPSHOT и READ COMMITTED средства резервирования дают для параллельных транзакций совершенно одинаковые результаты — см. далее.

## 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 относиться к глобальным временным таблицам (GTT), которые разрешено модифицировать 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, но не изменение данных, например, на компакт-дисках. Подробности см. в документе «Руководство администратора».

## Режим разрешения блокировок

В архитектуре клиент-сервер, когда с одной базой данных на сервере работает несколько клиентских приложений, блокировки всегда возникнут в том случае, когда один процесс вносит неподтвержденные изменения в строку таблицы или удаляет строку, не выполнив еще подтверждение удаления, а другой процесс пытается изменять или удалять эту же строку. Блокировки также могут возникнуть и в других ситуациях при использовании некоторых уровней изоляции транзакций.

Есть два режима разрешения блокировок: WAIT и NO WAIT.

В режиме WAIT (режим по умолчанию) при появлении конфликта с параллельными процессами, выполняющими конкурирующие обновления данных в той же базе данных, такая транзакция

будет ожидать завершения конкурирующей транзакции путем ее подтверждения (COMMIT) или отката (ROLLBACK). Иными словами, клиентское приложение будет переведено в режим ожидания до момента разрешения конфликта. Этот режим дает несколько отличные формы поведения в зависимости от уровня изоляции транзакций (см. далее). Если для режима WAIT задать предложение LOCK TIMEOUT, то ожидание будет продолжаться только указанное в этом предложении количество секунд. По истечении этого срока будет выдано сообщение об ошибке: "Lock time-out on wait transaction"(Истечение времени ожидания блокировки для транзакции WAIT).

Если установлен режим разрешения блокировок NO WAIT, то при появлении конфликта блокировки данная транзакция немедленно вызовет исключение базы данных.

## Уровень изоляции

Уровень изоляции запускаемой транзакции задается необязательным предложением ISOLATION LEVEL. Это самая важная характеристика транзакции, которая определяет ее основное поведение по отношению к другим одновременно выполняющимся транзакциям.

Различные уровни изоляции транзакций определяют поведение данного клиентского приложения, запустившего эту транзакцию, по отношению к другим параллельным процессам, выполняющимся на любом компьютере локальной сети, одновременно выполняющих чтение и/или изменение в той же базе данных, что и текущий процесс. Уровень изоляции является важнейшей характеристикой клиентского процесса, определяющей реакции других процессов на действия данного клиента, и на результаты обращений к базе данных этого клиента в зависимости от действий других параллельных процессов, выполняемых над данными этой базы данных.

В случае возникновения конфликта обновления данных (изменение существующих значений или удаление строк таблицы для двух и более параллельных транзакций) система управления базами данных выдает исключение базы данных соответствующего вида (см. приложение Б «Коды ошибок Ред База Данных»). При любом уровне изоляции транзакций в случае параллельно выполняющихся процессов конфликт блокировки возникнет всегда, когда одна транзакция пытается одновременно изменять или удалять запись, изменяемую другой параллельной транзакцией или удаленную другой параллельной транзакцией, когда эти изменения или удаления еще не подтверждены в соответствующей транзакции.

Конфликт блокировки также будет возникать, если одна транзакция удаляет строку главной таблицы в существующем в базе данных отношении главная-подчиненная (master-detail) или изменяет значение ключевого реквизита главной таблицы, не подтвердив выполненные изменения, а другая транзакция пытается изменить или удалить соответствующие данные в подчиненной таблице, те данные, которые связаны с главной таблицей обычным образом — внешний ключ / первичный (уникальный) ключ.

Конфликт возникнет и в том случае, если одна транзакция поместит в таблицу строку с конкретным значением первичного или уникального ключа (даже не подтвердив добавление данных), а другая параллельно выполняющаяся транзакция попытается поместить в эту таблицу строку с тем же значение первичного (уникального) ключа.

Существует три уровня изоляции транзакции: SNAPSHOT, SNAPSHOT TABLE STABILITY и уровень изоляции READ COMMITTED с двумя уточнениями (NO RECORD_VERSION и RECORD_VERSION).

### Уровень изоляции SNAPSHOT

Этот уровень изоляции транзакции является уровнем изоляции по умолчанию — когда при старте клиентом новой транзакции уровень ее изоляции в операторе SET TRANSACTION не указывается.

При уровне изоляции SNAPSHOT транзакция получает «мгновенный снимок» базы данных, соответствующий существующему на момент старта транзакции состоянию базы данных — количеству строк всех таблиц базы данных и их содержанию. Этот уровень изоляции означает, что этой транзакции видны лишь те изменения базы данных, которые были выполнены и подтверждены другими одновременно выполняющимися транзакциями на момент старта данной транзакции. Любые подтвержденные изменения, сделанные другими конкурирующими транзакциями, не будут видны в

такой транзакции в процессе ее активности без ее перезапуска. Чтобы увидеть эти изменения, нужно завершить транзакцию (подтвердить ее или выполнить полный откат, но не откат на точку сохранения — см. далее) и запустить транзакцию заново. Изменения данных, выполненные в контексте этой транзакции, подтвержденные или неподтвержденные, в этой транзакции видны.

Этот уровень изоляции позволяет в полном объеме получить описанный ранее эффект фантомных строк базы данных, когда отдельные записи любых таблиц были удалены другими параллельными процессами, а в данной транзакции они все еще видны как реально существующие, и эффект невоспроизводимого чтения, когда другие клиенты изменили существующие строки различных таблиц или добавили в таблицы новые строки, а данная транзакция видит лишь старое состояние базы данных.

Отрицательным свойством такого уровня изоляции транзакции является еще и тот факт, что если другой процесс выполнил изменение какой-либо строки таблицы базы данных и подтвердил это изменение, то попытка в данной транзакции позже внести изменения в эту же строку (уже после подтверждения изменений в другой транзакции) все равно вызовет исключение базы данных. Это во многих случаях очень часто приводит к конфликтам блокировки.

Этот уровень изоляции хорошо подходит для задач предметной области, где присутствует небольшое количество одновременно работающих клиентов, и когда нужно получить требуемые данные в условиях конкретного фиксированного состояния базы данных, то есть на момент старта клиентской транзакции.

При старте транзакции с таким уровнем изоляции можно запретить другим параллельно выполняющимся транзакциям вносить любые изменения в некоторые или во все таблицы базы данных, задав предложение резервирования для группы таблиц.

Для таблиц, указанных в предложении `RESERVING`, в параллельных транзакциях в зависимости от их уровня изоляции допустимы при различных способах их резервирования следующие варианты поведения:

- `SHARED READ` — не оказывает никакого влияния на выполнение параллельных транзакций;
- `SHARED WRITE` — на поведение параллельных транзакций с уровнями изоляции `SNAPSHOT` и `READ COMMITTED` не оказывает никакого влияния, для транзакций с уровнем изоляции `SNAPSHOT TABLE STABILITY` запрещает не только запись, но также и чтение данных из указанных таблиц;
- `PROTECTED READ` — допускает только чтение данных из резервируемых таблиц для параллельных транзакций с любым уровнем изоляции, попытка внесения изменений приводит к исключению базы данных;
- `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.

Если при попытке запуска транзакции, использующей резервирование таблиц, некоторые из указанных таблиц были изменены другими процессами и изменения не были подтверждены с помощью оператора `COMMIT` или отменены с использованием оператора `ROLLBACK`, то старт такой транзакции приведет к выдаче исключения базы данных.

## Уровень изоляции SNAPSHOT TABLE STABILITY

Уровень изоляции `SNAPSHOT TABLE STABILITY` похож на уровень изоляции `SNAPSHOT` только за тем лишь исключением, что при этом уровне изоляции другим параллельно выполняющимся транзакциям запрещено выполнять любые изменения во всех таблицах базы данных. Такая транзакция имеет исключительное, монопольное право на внесение любых изменений в данные базы данных.

При старте такой транзакции она так же, как и в случае использования уровня изоляции `SNAPSHOT`, получает мгновенный снимок состояния базы данных, который не изменяется, пока эта транзакция продолжает оставаться активной. Любые изменения в базе данных, выполненные в параллельных процессах, если им предоставлена такая возможность предложением резервирования, в данной транзакции не видны. Свои изменения транзакция видит. Если при старте транзакции с этим уровнем изоляции другой параллельный процесс выполнил любое изменение (добавление,

изменение, удаление) в какой-либо таблице базы данных и не подтвердил еще это изменение (если такая таблица не включена в список предложения резервирования, см. далее), то старт транзакции с режимом разрешения блокировок NO WAIT и с уровнем изоляции SNAPSHOT TABLE STABILITY приведет к исключению базы данных, транзакция не будет запущена. Если же для транзакции был задан режим разрешения блокировок WAIT, то транзакция будет в момент ее старта в подобной ситуации ожидать подтверждения или отмены выполненных другими транзакциями изменений.

Монопольный, исключительный режим для транзакции с данным уровнем изоляции можно отменить для некоторых или для всех таблиц базы данных, используя предложение резервирования RESERVING в операторе запуска транзакции SET TRANSACTION.

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.

Если при попытке запуска транзакции, использующей резервирование таблиц, некоторые из указанных в предложении резервирования таблиц были изменены другими параллельными процессами и эти изменения не были подтверждены с помощью оператора COMMIT, то старт такой транзакции приведет к выдаче исключения базы данных, если для транзакции был задан режим разрешения блокировок NO WAIT, иначе (при задании WAIT) транзакция перейдет в состояние ожидания, пока не будут подтверждены или отменены изменения соответствующих таблиц в других транзакциях.

## Уровень изоляции READ COMMITTED

Пожалуй, это наиболее часто используемый уровень изоляции для транзакций в случае использования многопользовательской системы обработки данных, когда много одновременно выполняющихся клиентских процессов работают с одной базой данных, находящейся на сервере, и должны постоянно иметь точные сведения об изменениях в таблицах базы данных, вносимых другими параллельными процессами.

Уровень изоляции READ COMMITTED позволяет в транзакции без ее перезапуска видеть все подтвержденные изменения данных базы данных, выполненные в других параллельных процессах. Неподтвержденные изменения не видны в транзакции и этого уровня изоляции.

При этом следует учитывать, что набор данных, полученный при первоначальном выполнении оператора SELECT, не будет отражать изменения, выполненные другими процессами, пока в рамках данной транзакции не будет выполнено переоткрытие набора данных, то есть пока не будет выполнен опять оператор SELECT в рамках активной транзакции READ COMMITTED без ее перезапуска.

Если в конфигурационном файле значение параметра ReadConsistency равно 1 (по умолчанию значение параметра равно 1), то при запуске такого запроса для него будет создаваться снимок версий, который будет использоваться запросом для чтения данных. Запрос будет видеть только те версии записей, которые были доступны на момент его старта.

По определению уровень изоляции READ COMMITTED не запрещает другим параллельным процессам вносить какие-либо изменения в таблицы текущей базы данных. При этом можно использовать предложение резервирования, чтобы защитить некоторые или все таблицы базы данных от изменения другими одновременно выполняющимися процессами.

Для таблиц, указанных в предложении RESERVING, в параллельных транзакциях в зависимости от их уровня изоляции допустимы при различных способах резервирования следующие варианты поведения (полностью соответствует средствам резервирования таблиц для уровня изоляции SNAPSHOT):

- SHARED READ — позволяет всем параллельным транзакциям независимо от их уровня изоляции не только читать, но и выполнять любые изменения в резервируемых таблицах (при уровне доступа READ WRITE);
- SHARED WRITE — для всех транзакций с уровнем доступа READ WRITE и с уровнями изоляции SNAPSHOT и READ COMMITTED позволяет читать 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 чтение данных из резервируемых таблиц для параллельных транзакций с любым уровнем изоляции;
- PROTECTED WRITE — для параллельных транзакций с уровнями изоляции SNAPSHOT и READ COMMITTED разрешает только чтение данных и запрещает запись в указанные в данном списке таблицы, для транзакций с уровнем изоляции SNAPSHOT TABLE STABILITY запрещает не только изменение данных, но и чтение данных из резервируемых таблиц.

Если при попытке запуска транзакции, использующей это резервирование таблиц, некоторые из указанных таблиц были изменены другими процессами и изменения не были подтверждены или отменены, то старт такой транзакции приведет к выдаче исключения базы данных, если для транзакции был задан режим разрешения блокировок NO WAIT, иначе транзакция перейдет в состояние ожидания, пока не будут подтверждены или отменены изменения соответствующих таблиц в других транзакциях.

При задании этого уровня изоляции транзакции можно указать предложение, модифицирующее условия появления исключений базы данных при блокировке:

```
[NO] RECORD_VERSION
```

Вариант NO RECORD_VERSION предполагается по умолчанию. Это означает, что при запуске такой транзакции в базе данных не должно быть измененных, добавленных или удаленных и неподтвержденных данных другими параллельными процессами. Если такие неподтвержденные изменения существуют, то при режиме разрешения блокировок WAIT транзакция перейдет в состояние ожидания, пока не будут подтверждены или отменены все изменения данных в параллельных процессах. При режиме разрешения блокировок NO WAIT попытка старта такой транзакции приведет к выдаче исключения базы данных.

При задании RECORD_VERSION транзакция при ее запуске читает последнюю подтвержденную версию записей таблиц, независимо от того, существуют ли другие измененные и еще не подтвержденные версии любых записей базы данных. В этом случае режим разрешения блокировок (WAIT или NO WAIT) никак не влияет на поведение транзакции при ее старте.

## 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 чаще всего завершаться успешно, а не откатываться.

Для транзакций, в рамках которых не выполняется никаких изменений, опция NO AUTO UNDO игнорируется.

### 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`).

## 10.2 Подтверждение транзакции

Чтобы подтвердить текущую транзакцию используется оператор `COMMIT` (см. листинг 10.3).

**Листинг 10.3. 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.

## 10.3 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 10.4. 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.

## 10.4 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 10.5.

**Листинг 10.5. Синтаксис оператора создания точки сохранения `SAVEPOINT`**

```
SAVEPOINT <имя точки сохранения>;
```

Имя точки сохранения — обычный идентификатор базы данных, который может содержать до 31 символа.

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 10.6.

**Листинг 10.6. 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 (см. листинг 10.7):

**Листинг 10.7. 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. Пользовательские переменные, заданные с помощью функции RDB$SET_CONTEXT() остаются неизменными.

Все последующие точки сохранения удаляются. 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.

## 10.5 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 подтвердила 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 11

# Хранимые процедуры, хранимые функции, триггеры и пакеты

В реляционных базах данных используются в первую очередь декларативные средства, когда пользователь системы в операторах DDL и DML описывает, что он хочет сделать с данными или метаданными базы данных, но не указывает, как это должно быть сделано. Кроме этих возможностей, можно использовать и императивные, процедурные, средства, когда пользователь точно шаг за шагом описывает выполняемые действия как с данными базы данных, так и с любыми другими внутренними данными.

Для этих целей используются программные элементы базы данных — хранимые процедуры, функции и триггеры.

Хранимые процедуры, функции и триггеры являются программами, которые хранятся в области метаданных базы данных. Они выполняются на стороне сервера, что во многих случаях позволяет сэкономить ресурсы системы и уменьшить сетевой трафик. К хранимым процедурам возможно обращение из хранимых процедур, триггеров и клиентских приложений. К хранимой функции могут обращаться хранимые процедуры, хранимые функции (в том числе и сама к себе), триггеры и клиентские программы. К триггерам напрямую обращение невозможно — они вызываются автоматически при наступлении конкретного события для таблицы (изменения данных) или события базы данных. Для каждого события таблицы существует две фазы — до наступления соответствующего события (`BEFORE`) и после наступления этого события (`AFTER`).

Для описания алгоритмов обработки данных в хранимых процедурах, функциях и в триггерах используется расширение языка SQL. Это расширение называется процедурным SQL (PSQL) или языком хранимых процедур, функций и триггеров. Язык содержит обычные операторы присваивания, операторы ветвления и операторы циклов. В триггерах могут применяться специфические контекстные переменные.

## 11.1 Язык хранимых процедур, функций и триггеров

Этот язык является обычным языком программирования, который содержит все основные конструкции классических языков программирования. Кроме того, в нем присутствуют несколько модифицированные операторы добавления, изменения, удаления и выборки существующих данных из таблиц базы данных (`INSERT, UPDATE, DELETE` и `SELECT`). В языке нельзя выполнять какие-либо изменения метаданных. Нельзя также использовать оператор `EXECUTE BLOCK`.

В хранимых процедурах, функциях и триггерах можно использовать средства оповещения клиентов о некоторых событиях (`events`), возможности выдавать пользовательские исключения (`exceptions`). Недопустимо выполнять операции соединения с базой данных, нельзя манипулировать транзакциями, включая старт транзакции, создание точек сохранения, возврата на точки сохранения, подтверждения или отмены транзакций. Хранимая процедура, функция и триггер выполняются в контексте той транзакции, при которой была явно вызвана хранимая процедура, хранимая функция или выполнялась операция манипулирования данными в базе данных, в результате чего был автоматически запущен триггер. Если триггер вызывается при соединении с базой данных и отсоединении от нее, то для него запускается транзакция по умолчанию.

В синтаксисе создания хранимых процедур, функций и триггеров можно выделить заголовок и тело. Заголовок содержит имя программного объекта, описание локальных переменных. Для триггеров в заголовке указывается событие базы данных и фаза, при которой автоматически вызывается триггер. В заголовке хранимой процедуры можно указать входные и выходные параметры.

В заголовке хранимой функции можно указать входные параметры и тип выходного результата. Тело хранимой процедуры, функции или триггера представляет собой блок операторов, содержащий описание выполняемых программой действий. Блок операторов заключается в операторные скобки BEGIN и END. В самих программах возможно присутствие произвольного количества блоков, как последовательных, так и вложенных друг в друга.

## Использование оператора SET TERM

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 в конце текста триггера, функции или хранимой процедуры. 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
... <Текст триггера>
END ^
...
[<Другие создаваемые триггеры, хранимые функции или хранимые процедуры>]
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.

## Внутренние переменные

В триггерах, хранимых функциях и хранимых процедурах можно использовать локальные переменные. Для описания одной локальной переменной используется оператор DECLARE VARIABLE. Его упрощенный синтаксис представлен в листинге 11.1.

**Листинг 11.1. Несколько упрощенный синтаксис оператора объявления локальной переменной 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>}
```

Ключевое слово VARIABLE можно опустить. В одном операторе можно объявить только одну локальную переменную. В триггере, хранимой функции и процедуре можно объявлять произвольное количество локальных переменных, используя для каждой переменной отдельный оператор DECLARE VARIABLE.

Имя локальной переменной должно быть уникальным среди имен локальных переменных, входных и выходных параметров хранимой процедуры в пределах данного программного объекта.

Типом данных может быть любой тип данных, используемый в SQL.

Вместо типа данных можно указать имя ранее созданного домена. В этом случае 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.

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 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.

Полный синтаксис оператора DECLARE будет описан далее в этой главе.

В хранимых процедурах могут быть также описаны входные и выходные параметры, в хранимых функциях только входные параметры. Список входных параметров записывается после имени хранимой процедуры или функции и заключается в круглые скобки:

```
<входной/выходной 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> }
```

Круглые скобки, в которые заключается список входных параметров, по правилам синтаксиса SQL могут быть опущены.

Если в описании входного параметра задана ссылка на домен, то входному параметру присваиваются все характеристики этого домена за исключением значения по умолчанию (предложение DEFAULT в описании домена). При обращении к хранимой процедуре и функции всегда нужно задавать значения всех входных параметров.

Выходные параметры описываются в предложении RETURNS:

```
RETURNS (<выходной параметр> [, <выходной параметр> ...])
```

Описание выходного параметра соответствует описанию входного параметра. Если выходной

параметр ссылается на домен, то ему также присваивается и значение по умолчанию, в отличие от входного параметра.

Локальные переменные триггеров, функций и процедур, входные и выходные параметры, используемые только в хранимых процедурах и функциях, являются внутренними переменными. Имена внутренних переменных могут совпадать с именами столбцов используемых таблиц базы данных. Это не вызовет проблемы двусмысленности имен. При использовании внутренних переменных в операторах `SELECT, INSERT, UPDATE` или `DELETE` именам внутренних переменных всегда должно предшествовать двоеточие, чтобы не спутать их с именами столбцов таблицы. Во всех остальных случаях в любых других операторах имена внутренних переменных записываются обычным образом без двоеточия.

### Предварительно определенные литералы и контекстные переменные

В триггерах, в хранимых процедурах и функциях могут быть использованы предварительно определенные литералы и контекстные переменные, часть из которых может применяться в любых операторах SQL, другая же часть может быть использована только в триггерах.

Подробные описания характеристик и порядка использования предварительно определенных литералов и контекстных переменных см. в главе 2 «Типы данных Ред База Данных» и в приложении Ж «Контекстные переменные».

Следующие контекстные переменные могут быть использованы только в PSQL.

Контекстная переменная `ROW_COUNT` типа `INTEGER` может быть использована в триггерах и в хранимых процедурах. Она возвращает общее количество строк, которые были прочитаны, добавлены, изменены или удалены в процессе выполнения последнего оператора SQL. Чаще всего эта контекстная переменная используется после оператора `SELECT` или после оператора `FETCH`, читающего очередную запись из таблицы, заданной объявленным курсором (см. далее в этой главе); в этом случае она содержит количество считанных данным оператором строк. Обычно используется для определения завершения считывания данных из таблицы (представления), определенной объявленным внутренним курсором. Соответствующий пример будет рассмотрен далее.

Контекстные переменные `SQLCODE` и `GDSCODE` типа `INTEGER` позволяют получить значения соответствующих кодов ошибок базы данных, когда последняя операция обращения к базе данных завершилась с ошибкой. Могут использоваться в хранимых процедурах и функциях или триггерах и только в блоках обработки ошибок базы данных `WHEN-DO`. За пределами таких блоков эти переменные имеют нулевое значение.

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 использоваться только в триггерах. Они применяются в тех триггерах, которые выполняются сразу для нескольких событий одной таблицы.

## 11.2 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 или хранимой процедуры и функции обнаруживается соответствующая ошибка в базе данных (обычно на содержательном, а не на формальном уровне), то можно вызвать пользовательское исключение.

В PSQL существуют средства обработки вызванных пользовательских исключений — см. далее в этой главе подраздел 11.4 Обработка ошибочных ситуаций.

Для создания пользовательского исключения используется оператор `CREATE EXCEPTION`. Его синтаксис показан в листинге 11.2.

**Листинг 11.2. Синтаксис оператора создания пользовательского исключения CREATE EXCEPTION**

```
CREATE EXCEPTION <имя исключения> '<текст сообщения>';
```

Имя исключения может содержать до 31 символа 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.

Для изменения текста сообщения существующего пользовательского исключения используется оператор `ALTER EXCEPTION`. Синтаксис оператора см. в листинге 11.3:

**Листинг 11.3. 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`.

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 11.4.

**Листинг 11.4. Синтаксис оператора CREATE OR ALTER EXCEPTION**

```
CREATE OR ALTER EXCEPTION <имя исключения> '<текст сообщения>';
```

Оператор `RECREATE EXCEPTION` создает новое пользовательское исключение, если оно отсутствует в базе данных, или пересоздает существующее. Синтаксис оператора см. в листинге 11.5.

**Листинг 11.5. Синтаксис оператора RECREATE EXCEPTION**

```
RECREATE EXCEPTION <имя исключения> '<текст сообщения>';
```

Для удаления существующего пользовательского исключения используется оператор `DROP EXCEPTION` (листинг 11.6).

**Листинг 11.6. 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.

Для вызова в триггере или в хранимой процедуре и функции пользовательского исключения нужно выполнить оператор `EXCEPTION` (листинг 11.7).

**Листинг 11.7. 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 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.

**Пример.** В базе данных существует таблица PEOPLE. В ней для каждого человека можно указать код его матери и код его отца. Структура таблицы описана в главе 8 в листинге 8.3. Если при помещении в эту таблицу новой строки указать неправильный код, например, код матери, то можно, предварительно создав соответствующее пользовательское исключение, вызвать это исключение.

**Пример 11.1**

```
CREATE EXCEPTION NO_MOTHER
'В базе данных отсутствует запись, соответствующая матери человека по имени @1';
...
INSERT INTO PEOPLE (..., CODMOTHER, ...)
VALUES (..., '0', ...);
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, или хранимая процедура, или хранимая функция) в блоке обработки ошибок WHEN-DO выдаст пользовательское исключение с именем NO_MOTHER и с текстом 'В базе данных отсутствует запись, соответствующая матери человека'. Если в данной программе не задана обработка подобного исключения, то работа программы завершается. Новая запись не будет помещена в базу данных.

Пользовательские исключения могут быть перехвачены и обработаны в триггере, в хранимой функции или в хранимой процедуре, при использовании оператора WHEN-DO (см. подраздел 11.4 Обработка ошибочных ситуаций в этой главе).

## 11.3 События 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 измененное состояние данных.

Для отправки сообщения (события) используется оператор `POST_EVENT`. Его синтаксис представлен в листинге 11.8.

**Листинг 11.8. Синтаксис оператора передачи события `POST_EVENT`**

```
POST_EVENT {
   '<имя сообщения>'
  | <имя столбца>
  | :<внутренняя переменная> };
```

Отправляемый текст может быть задан именем сообщения, заключенным в апострофы, именем столбца таблицы, который содержит соответствующую строку, или именем внутренней переменной строкового типа (локальная переменная, входной или выходной параметр хранимой процедуры), куда помещено нужное значение. Перед именем внутренней переменной должно быть помещено двоеточие, чтобы отличить имя переменной от имени столбца таблицы.

## 11.4 Операторы языка хранимых процедур, функций и триггеров

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.

### Объявление локальных переменных и курсоров

В триггерах, хранимых процедурах и функциях можно объявлять локальные переменные и курсоры. Синтаксис оператора представлен в листинге 11.9.

**Листинг 11.9. Синтаксис оператора объявления локальной переменной и курсора `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.

Подробно работа с курсорами описана далее в этой главе.

## Объявление подпроцедуры

В хранимых функциях и хранимых процедурах можно объявлять подпроцедуры. Синтаксис оператора представлен в листинге 11.10.

**Листинг 11.10. 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. Кроме того, подпрограмма не может вызывать себя рекурсивно. Это может быть разрешено в будущем.

## Объявление подфункции

В хранимых функциях и хранимых процедурах можно объявлять подфункции. Синтаксис оператора представлен в листинге 11.11.

**Листинг 11.11. 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. Кроме того, подпрограмма не может вызывать себя рекурсивно. Это может быть разрешено в будущем.

## Операция присваивания

Синтаксис операции присваивания значения внутренней переменной имеет следующий вид:

```
<имя переменной> = <выражение>;
```

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, определенным пользователем (UDF).

Синтаксис выражения приведен в листинге 11.12.

**Листинг 11.12. Синтаксис выражения**

```
<выражение> ::= {
    <внутренняя переменная>
  | <контекстная переменная>
  | <литерал>
  | <арифметическое выражение>
  | <строковое выражение>
  | <логическое выражение>
  | NEXT VALUE FOR <имя генератора>
  | <обычная встроенная функция> (<параметры>)
  | <агрегатная функция в операторе SELECT>
  | <функция UDF> [(<параметр> [, <параметр>]...)]
  | NULL }
```

Литерал — это числовая константа, строковая константа, заключенная в апострофы, литерал даты или времени, предварительно определенный литерал, контекстная переменная.

Арифметическое выражение содержит четыре арифметические операции — сложение (+), вычитание (–), умножение (*) и деление (/).

Строковое выражение представлено одной строковой операцией конкатенации (||) — соединения двух строк в одну.

Логическое выражение может содержать операцию отрицания (NOT), дизъюнкции (OR) и конъюнкции (AND). В языке не существует логических констант.

Формально арифметическое выражение определяется следующим образом (см. листинг 11.13).

**Листинг 11.13. Синтаксис арифметического выражения**

```
<арифметическое выражение> ::= {
    <числовой литерал>
  | (<арифметическое выражение>)
  | <арифметическое выражение> + <арифметическое выражение>
  | <арифметическое выражение> - <арифметическое выражение>
  | <арифметическое выражение> * <арифметическое выражение>
  | <арифметическое выражение> / <арифметическое выражение>
  | <выражение типа DATE> - <число>
  | <выражение типа DATE> + <число>
  | <выражение типа DATE> - <выражение типа DATE>
  | <выражение типа TIME> - <число>
  | <выражение типа TIME> + <число> }
```

Числовой литерал — число с фиксированной или плавающей точкой. Подробнее описание числовых литералов, арифметических операций над числами, датами и временем см. в главе 2 «Типы данных Ред База Данных».

Синтаксис строкового выражения представлен в листинге 11.14.

**Листинг 11.14. Синтаксис строкового выражения**

```
<строковое выражение> ::= {
    <строковый литерал>
  | (<строковое выражение>)
  | <строковое выражение> || <строковое выражение> }
```

Строковый литерал — последовательность любых символов, заключенная в апострофы.

Синтаксис логического выражения представлен в листинге 11.15.

**Листинг 11.15. Синтаксис логического выражения**

```
<логическое выражение> ::= {
    <операция сравнения>
  | (<логическое выражение>)
  | NOT <логическое выражение>
  | <логическое выражение> AND <логическое выражение>
  | <логическое выражение> OR <логическое выражение> }
```

Конструкция NEXT VALUE FOR <имя генератора> является аналогом функции GEN_ID (<имя генератора>, 1). Значение указанного генератора увеличивается на единицу, и конструкция возвращает новое значение.

Существует два типа встроенных функций — обычные встроенные функции и агрегатные функции в операторе SELECT.

Обычная встроенная функция — это функция, работающая с одним или более параметрами. Функция возвращает ровно одно значение. Агрегатные функции в операторе SELECT — особые функции, определенные в языке SQL Ред База Данных. Они работают не с одним фиксированным набором параметров, а с группой значений, полученных при выполнении определенного оператора SELECT из таблицы базы данных. Агрегатные функции используются внутри списка выбора этого оператора SELECT.

Встроенные функции подробно описаны в приложении Е «Функции».

Выражением также может быть обращение к функции, определенной пользователем (User

Defined Function, UDF — см. приложение Г).

В качестве выражения может быть также указано пустое значение NULL.

## Оператор IF-THEN-ELSE

Для выполнения ветвления процесса обработки данных в PSQL используется оператор IF-THEN-ELSE. Его синтаксис представлен в листинге 11.16.

**Листинг 11.16. Синтаксис оператора IF-THEN-ELSE**

```
IF (<условие>)
THEN <составной оператор>
[ELSE <составной оператор>];
```

Условием является обычное условие, принятое в SQL, 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 это ключевое слово присутствует. Условие всегда заключается в круглые скобки.

Составной оператор — это одиночный оператор или блок операторов, заключенных в операторные скобки BEGIN и 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.

**Пример 11.2**

```
IF (INPUT_PASSWORD <> STORED_PASSWORD) THEN
   EXCEPTION WRONG_PASSWORD;
...
```

## Оператор WHILE-DO

Оператор WHILE-DO позволяет организовать в PSQL обычный цикл. Синтаксис оператора представлен в листинге 11.17.

**Листинг 11.17. Синтаксис оператора WHILE-DO**

```
WHILE (<условие>) DO
   <составной оператор>
```

Составной оператор будет выполняться в цикле, пока условие возвращает значение TRUE.

Помимо оператора WHILE-DO, существуют другие операторы цикла — FOR SELECT-DO и FOR EXECUTE STATEMENT. Описание операторов см. далее в этой главе.

Циклы могут быть вложенными, глубина вложения не ограничена.

В следующем фрагменте хранимой процедуры (Пример 11.3) 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.

**Пример 11.3**

```
...
```

```
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;
...
```

## Операторы перехода

В PSQL не существует оператора `GO TO`, выполняющего переход на указанную метку в программном тексте, что соответствует правилам структурного программирования. При этом есть операторы, позволяющие выйти из циклов или перейти на начало этого же или другого цикла (`LEAVE`), перейти на финальный оператор `END (EXIT)`, временно приостановить выполнение хранимой процедуры для передачи вызвавшей программе полученных данных (`SUSPEND`) и досрочно начать новую итерацию цикла (`CONTINUE`).

### Оператор EXIT

Оператор `EXIT` позволяет из любой точки триггера или хранимой процедуры, функции перейти на конечный оператор `END`, то есть завершить выполнение программы (листинг 11.18).

**Листинг 11.18. Синтаксис оператора `EXIT`**

```
EXIT [<метка>];
```

### Оператор LEAVE

Этот оператор осуществляет выход из цикла `WHILE-DO` независимо от выполнения условия в предложении `WHILE`. Если же в операторе указана метка, то осуществляется переход на начало другого (или того же самого) цикла. Синтаксис оператора представлен в листинге 11.19.

**Листинг 11.19. Синтаксис оператора `LEAVE`**

```
LEAVE [<метка>];
```

Если в операторе не указана метка, то оператор просто осуществляет выход из текущего цикла `WHILE`. Пример 11.2 можно изменить, используя оператор `LEAVE`:

**Пример 11.4**

```
...
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.

Если в операторе LEAVE указана метка, то это должна быть метка, относящаяся к оператору WHILE-DO, к оператору FOR SELECT-DO или к оператору FOR EXECUTE STATEMENT. При выполнении такого оператора LEAVE происходит переход к выполнению соответствующего циклического оператора.

## Оператор SUSPEND

Оператор временно приостанавливает выполнение хранимой процедуры выбора (процедуры, которая чаще всего содержит оператор SELECT, выбирающий множество строк таблицы, обращение к такой процедуре также выполняется при помощи оператора SELECT — см. далее) и передает вызвавшей программе значения выходных параметров. Когда вызвавшая программа, обработав очередную строку, выполняет после этого (явно или неявно) оператор FETCH, работа процедуры возобновляется с оператора, следующего непосредственно за оператором SUSPEND (листинг 11.20).

**Листинг 11.20. Синтаксис оператора SUSPEND**

```
SUSPEND;
```

Если оператор SUSPEND выдается в выполняемой хранимой процедуре (в процедуре, которая вызывается оператором EXECUTE PROCEDURE), то это равносильно выполнению оператора EXIT, в результате чего полностью завершается работа процедуры.

## Оператор 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 11.21. Синтаксис оператора CONTINUE**

```
CONTINUE [<метка>];
```

## Оператор EXECUTE PROCEDURE

Триггеры, хранимые процедуры и функции могут вызывать хранимые процедуры при использовании оператора EXECUTE PROCEDURE. Оператор также может быть использован в подмножестве языка DML и в утилите isql. Синтаксис оператора приведен в листинге 11.22.

**Листинг 11.22. Синтаксис оператора вызова процедуры 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 этой процедуры. Если процедура получает параметры, то список входных параметров в операторе EXECUTE PROCEDURE является обязательным. При этом требуется полное соответствие количества передаваемых процедуре параметров и их типов данных описанным в процедуре входным параметрам.

Если оператор EXECUTE PROCEDURE вызывается из утилиты командной строки isql, то нельзя использовать предложение RETURNING_VALUES.

Примеры создания и вызова хранимых процедур см. далее в этой главе в разделе 11.6 Работа с хранимыми процедурами.

## Обычные операторы обращения к базе данных

В хранимых процедурах, функциях и триггерах можно использовать обычные операторы, выполняющие выборку данных (SELECT), добавление (INSERT), изменение (UPDATE) и удаление (DELETE) данных. Синтаксис таких операторов см. в соответствующих главах этого документа. Отличительной особенностью использования этих операторов в языке хранимых процедур, функций и триггеров является только то, что в любом внутреннем предложении таких операторов обращения к базе данных (в частности, в предложении WHERE) в качестве значения может использоваться имя внутренней переменной (локальной переменной, входного или выходного параметра хранимой процедуры), перед которым обязательно должно помещаться двоеточие, чтобы не спутать внутренние переменные с именами столбцов таблицы.

Например, чтобы выполнить удаление регионов страны, чей код задан переменной CODCOUNTRY (имя этой переменной совпадает с именем столбца в таблице регионов REGION), в хранимой процедуре, функции или в триггере нужно выполнить следующий оператор DELETE:

```
DELETE FROM REGION
WHERE CODCOUNTRY = :CODCOUNTRY;
```

Чтобы изменить код страны у всех регионов, относящихся к одной конкретной стране, чей код хранится во внутренней переменной CODCOUNTRY, нужно выполнить оператор UPDATE:

```
UPDATE REGION
SET CODCOUNTRY = :NEWCODCOUNTRY
WHERE CODCOUNTRY = :CODCOUNTRY;
```

Следующий оператор INSERT добавляет в таблицу стран COUNTRY новую страну. Значения для столбцов новой записи выбираются из внутренних переменных:

```
INSERT INTO COUNTRY (CODCOUNTRY, NAME, FULLNAME, CAPITAL)
VALUES (:CODCOUNTRY, :NAME, :FULLNAME, :CAPITAL);
```

В операторе SELECT, выбирающем данные из таблицы, представления или хранимой процедуры выбора, помимо остальных предложений должно присутствовать предложение INTO, в котором

перечисляются имена внутренних переменных, куда будут помещаться значения столбцов считанной строки таблицы. Именам внутренних переменных предшествует двоеточие, чтобы не спутать их с именами столбцов таблицы базы данных. Предложение INTO должно быть последним в этом операторе.

Например, следующий оператор SELECT выбирает строку из таблицы стран COUNTRY с кодом страны, находящимся во внутренней переменной CODCOUNTRY, и помещает краткое название страны и полное название страны в две внутренние переменные NAME и FULLNAME, которые имеют те же имена, что и столбцы таблицы:

```
SELECT
   NAME,
   FULLNAME
FROM COUNTRY
WHERE CODCOUNTRY = :CODCOUNTRY
INTO :NAME, :FULLNAME;
```

В PSQL существует циклический оператор FOR SELECT-DO, который обычно используется в хранимых процедурах для получения данных из таблиц, представлений или хранимых процедур выбора базы данных.

## Оператор FOR SELECT-DO

Оператор FOR SELECT-DO является оператором цикла, выбирающим строки из таблицы, представления, хранимой процедуры выбора. Синтаксис оператора представлен в листинге 11.23.

**Листинг 11.23. Синтаксис оператора цикла FOR SELECT-DO**

```
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, который может быть одним оператором или блоком операторов, заключенных в операторные скобки BEGIN и END. Оператор SELECT должен содержать предложение INTO, которое располагается в конце оператора. Цикл повторяется, пока не будут прочитаны все строки. После этого происходит выход из цикла. Цикл также может быть завершен и раньше 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.

В главе 9 в листинге 9.6 был показан пример создания представления VIEW_RUSSIA2, выбирающего все регионы страны Россия. Следующий пример показывает возможность использования этого представления в цикле FOR SELECT-DO.

**Пример 11.5**

```
FOR SELECT
   CODCOUNTRY, CODREGION, NAMEREG, CENTER
FROM VIEW_RUSSIA2
INTO
```

```
   :CODCOUNTRY, :CODREGION, :NAMEREG, :CENTER
DO
   BEGIN
     EXECUTE PROCEDURE PROC_N (CODCOUNTRY, CODREGION, NAMEREG, CENTER);
     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.

## Оператор FOR EXECUTE STATEMENT

Оператор `FOR EXECUTE STATEMENT` является оператором цикла. Синтаксис оператора представлен в листинге 11.24.

**Листинг 11.24. Синтаксис оператора цикла `FOR EXECUTE STATEMENT`**

```
[FOR] EXECUTE STATEMENT <строковое выражение>
   [ON EXTERNAL [DATA SOURCE] <строка соединения> [READ ONLY | READ WRITE]]
   [WITH {AUTONOMOUS TRANSACTION [READ ONLY | READ WRITE] | 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> ::= <имя параметра>:=<выражение>
                            [, <имя параметра>:=<выражение> ...]

<позиционные параметры> ::= <выражение> [, <выражение> ...]
```

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 Приложении З.

Пример использования оператора `FOR EXECUTE STATEMENT` представлен в Примере 11.6. В следующем фрагменте хранимой процедуры выбора выполняются те же действия, что и в операторе

FOR SELECT-DO, показанном в Примере 11.5.

**Пример 11.6**

```
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);
     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.

## Использование курсоров

В триггерах, хранимых процедурах и функциях существует возможность использования курсоров — локальных переменных, связанных с оператором SELECT, который возвращает набор данных, полученный из таблицы (таблиц) базы данных или из представления. Получаемый при помощи курсора набор данных по умолчанию является однонаправленным, то есть можно последовательно читать из него данные, начиная с первой, и последовательно выбирать данные вплоть до получения последней строки.

Для использования этих средств необходимо в хранимой процедуре, функции или в триггере объявить локальную переменную определенного вида — курсор. В процессе выполнения хранимой процедуры, функции или триггера нужно открыть этот курсор (оператор OPEN). После чего можно читать данные при использовании этого курсора (оператор FETCH). После считывания всех записей курсор нужно закрыть (оператор CLOSE). Для определения того, что считаны все строки, полученные с использованием курсора, можно использовать контекстную переменную COUNT_ROW.

Для объявления локальной переменной – курсора используется следующий вариант синтаксиса оператора DECLARE VARIABLE (листинг 11.25).

**Листинг 11.25. Синтаксис оператора объявления курсора DECLARE VARIABLE**

```
DECLARE [VARIABLE] <имя курсора> [SCROLL | NO SCROLL]
   CURSOR FOR (<оператор SELECT>);
```

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.

Оператор SELECT задает выборку данных из таблицы (таблиц) базы данных или из представления. Это может быть сколь угодно сложный оператор, включающий объединения и соединения

таблиц при наличии любых условий выборки данных. Собственно выборка данных осуществляется при выполнении оператора OPEN для этого курсора.

**Листинг 11.26. Синтаксис оператора открытия курсора OPEN**

```
OPEN <имя курсора>;
```

Этот оператор выполняет оператор SELECT, заданный при объявлении курсора. После выполнения этого оператора возможно получение данных из набора данных указанного курсора.

Данные очередной строки таблицы (представления) при использовании курсора получаются при выполнении оператора FETCH для этого курсора. Синтаксис оператора FETCH представлен в листинге 11.27.

**Листинг 11.27. Синтаксис оператора чтения очередной строки из курсора 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 лишь в том случае, если для полученной строки в дальнейшем будет использован только 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 предло-

жением WHERE CURRENT OF) установит NULL для значений полей переменной курсора для последующих чтений.

Для проверки того, что исходные записи для набора данных исчерпаны, используется контекстная переменная ROW_COUNT, которая возвращает количество считанных оператором FETCH строк. Если произошло чтение очередной записи из набора данных, то ROW_COUNT равняется единице, иначе при достижении конца исходных данных эта контекстная переменная будет равна нулю.

После завершения работы с данными, полученными при помощи курсора, необходимо закрыть курсор, используя оператор CLOSE. Вообще говоря, курсор явно можно и не закрывать. Он автоматически будет закрыт по завершении выполнения хранимой процедуры, функции, триггера. Синтаксис оператора приведен в листинге 11.28.

**Листинг 11.28. Синтаксис оператора закрытия курсора CLOSE**

```
CLOSE <имя курсора>;
```

Оператор закрывает курсор и освобождает все ресурсы вычислительной системы, связанные с этим курсором и полученным набором данных.

**Пример использования курсора.** Следующий фрагмент хранимой процедуры (см. Пример 11.7) выполняет выборку данных из таблицы стран. Результат полностью соответствует тем, которые получаются в случае выполнения операторов, описанных в примерах 11.5 и 11.6.

**Пример 11.7**

```
DECLARE VARIABLE NEW_CURSOR
   CURSOR FOR (SELECT
                 CODCOUNTRY, CODREGION, NAMEREG, CENTER
               FROM VIEW_RUSSIA2);
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 ^
```

## Оператор IN AUTONOMOUS TRANSACTION

Оператор IN AUTONOMOUS TRANSACTION позволяет выполнить оператор или блок операторов в автономной транзакции. Синтаксис оператора представлен в листинге 11.29)

**Листинг 11.29. Синтаксис оператора IN AUTONOMOUS TRANSACTION**

```
IN AUTONOMOUS TRANSACTION DO <оператор/блок операторов>
```

Код, работающий в автономной транзакции, будет подтверждаться сразу же после успешного завершения независимо от состояния родительской транзакции. Это бывает нужно, когда определённые 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.

## Обработка ошибочных ситуаций

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 11.30.

**Листинг 11.30. Синтаксис оператора обработки ошибок базы данных или пользовательских исключений 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> }
```

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 попытке поместить в таблицу строку, имеющую дублирующее значение первичного или уникального ключа.

**Пример 11.8**

```
INSERT INTO COUNTRY (CODCOUNTRY) VALUES ('USA');
WHEN SQLCODE -803
DO ... ;
```

Однако подобное значение SQLCODE будет сформировано и при попытке поместить в базу данных новой строки, создающей дублирующее значение построенного пользователем индекса (см. приложение Б). Чтобы более тонко определить данную конкретную ошибочную ситуацию, следует использовать значение не код SQLCODE, а GDSCODE.

**Пример 11.9**

```
INSERT INTO COUNTRY (CODCOUNTRY) VALUES ('USA');
WHEN GDSCODE -335544665
DO ... ;
```

Этот обработчик срабатывает уже только на дублированное значение первичного ключа помещаемой в базу данных новой записи.

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 оператора `WHEN-DO` с ключевым словом `EXCEPTION`. В примере 11.1 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 приложении Б «Коды ошибок Ред База Данных». Работу с пользовательскими исключениями см. в разделе 11.2 Пользовательские исключения данной главы.

# 11.5 Работа с триггерами

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 два 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.

## 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).

Существует возможность создавать триггеры, вызываемые автоматически для одной таблицы (представления), для одной фазы и одного события, а также для одной фазы и нескольких событий. Контекстные переменные INSERTING, UPDATING и DELETING логического типа могут быть использованы в теле триггера для определения события, которое вызвало срабатывание триггера.

Если для одной таблицы (представления), одного события и одной фазы существует несколько триггеров, то можно задать порядок их выполнения, указав позицию триггера в этой цепочке.

Для триггеров существуют специфические контекстные переменные OLD.<columname> и NEW.<columname>. Более правильное название этих ключевых слов — префиксы имен столбцов. 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 для всех видов триггеров является переменной только для чтения. Она недоступна в триггерах, вызываемых при добавлении данных, независимо от фазы события.

Контекстная переменная NEW в триггерах для фазы события после (AFTER) также является переменной только для чтения. Она недоступна в триггерах для события удаления данных.

## Триггеры на события базы данных

Триггер, связанный с событиями базы данных, может вызываться при следующих событиях:

- при соединении с базой данных (CONNECT);
  перед выполнением триггера автоматически запускается транзакция по умолчанию;
- при отсоединении от базы данных (DISCONNECT);
  перед выполнением триггера запускается транзакция по умолчанию;
- при старте транзакции (TRANSACTION START);
  триггер выполняется в контексте текущей транзакции;
- при подтверждении транзакции (TRANSACTION COMMIT);
  триггер выполняется в контексте текущей транзакции;
- при отмене транзакции (TRANSACTION ROLLBACK);
  триггер выполняется в контексте текущей транзакции.

Триггера на события CONNECT и DISCONNECT выполняются в специально созданной для этого транзакции. Если при обработке триггера не было вызвано исключение, то транзакция подтверждается. Не перехваченные исключения откатят транзакцию и: в случае триггера на событие CONNECT соединение разрывается, а исключения возвращается клиенту; для триггера на событие DISCONNECT соединение разрывается, как это и предусмотрено, но исключения не возвращается клиенту.

Триггера на событие TRANSACTION срабатывают при старте транзакции, её подтверждении или отмене. Не перехваченные исключения обрабатываются в зависимости от типа события TRANSACTION: для события START исключение возвращается клиенту, а транзакция отменяется; для события COMMIT исключение возвращается клиенту, действия, выполненные триггером, и транзакция отменяются; для события ROLLBACK исключение не возвращается клиенту, а транзакция, как и предусмотрено, отменяется.

В случае двухфазных транзакций триггеры на событие TRANSACTION START срабатывают в фазе подготовки (prepare), а не в фазе commit.

## 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.

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.

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.

## Создание триггера

Для создания триггера используется оператор CREATE TRIGGER, синтаксис которого представлен в листинге 11.31.

**Листинг 11.31. 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
   [<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
 | 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>
                 | <объявление 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`.

Имя триггера может содержать до 31 символа и должно быть уникальным среди имен всех триггеров базы данных.

Триггер может быть создан как для таблицы (представления) базы данных, для события изменения метаданных, так и для события базы данных.

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.

Если триггер создается для таблицы (представления), то для него указывается событие и фаза события.

Ключевое слово `BEFORE` означает, что триггер вызывается до наступления соответствующего события (событий, если их указано несколько), `AFTER` — после наступления события (событий).

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 позиции для триггеров не заданы или несколько триггеров имеют одно и то же значение позиции, то такие триггеры будут выполняться в алфавитном порядке их имен.

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.

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.

В теле триггера может быть описано произвольное количество локальных переменных, именованных курсоров и подпрограммы (подпроцедуры и подфункции). Синтаксис оператора объявления локальной переменной и курсора см. в листинге 11.9 в этой главе.

После описания локальных переменных и подпрограмм в теле триггера следует блок опера-

торов, заключенных в операторные скобки BEGIN и END.

## Изменение триггера

Для изменения заголовка и/или тела существующего триггера используется оператор ALTER TRIGGER, синтаксис которого представлен в листинге 11.32.

**Листинг 11.32. 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}]
[{EXTERNAL NAME '<внешний модуль>' ENGINE <имя движка>} |
{
  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 триггером 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 только изменить его активность, позицию и выполняемые действия. 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.

## 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 11.33. 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
   [<объявление> [<объявление> ...] ]
  BEGIN
   <блок операторов>
  END }
```

Синтаксис этого оператора соответствует синтаксису оператора `CREATE TRIGGER`.

## Удаление триггера

Для удаления существующего триггера используется оператор `DROP TRIGGER`, синтаксис которого представлен в листинге 11.34.

**Листинг 11.34. 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.

## 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 11.35. 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
   [<объявление> [<объявление> ...] ]
  BEGIN
   <блок операторов>
  END }
```

Синтаксис этого оператора соответствует синтаксису оператора `CREATE TRIGGER`.

# 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.

## 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:

**Пример 11.10**

```
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);
```

## 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:

**Пример 11.11**

```
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).

## 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 TBU_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.

**Пример 11.12**

```
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.

**Пример 11.13**

```
CREATE OR ALTER TRIGGER TAD_COUNTRY
```

```
   FOR COUNTRY AFTER DELETE
AS BEGIN
   DELETE FROM REGION
   WHERE REGION.CODCOUNTRY = OLD.CODCOUNTRY;
END ^
```

## Пример триггера, создающего запись истории окладов

В базе данных EMPLOYEE, являющейся демонстрационной базой данных InterBase, присутствуют таблица EMPLOYEE, описывающая сотрудников организации, и таблица SALARY_HISTORY, хранящая историю окладов сотрудников. Операторы создания этих таблиц представлены в примере 11.14.

**Пример 11.14**

```
CREATE TABLE EMPLOYEE (
   emp_no EMPNO NOT NULL PRIMARY KEY,
   first_name FIRSTNAME NOT NULL,
   last_name LASTNAME NOT NULL,
   phone_ext VARCHAR(4),
   hire_date DATE DEFAULT 'NOW' NOT NULL,
   dept_no DEPTNO,
   job_code JOBCODE NOT NULL,
   job_grade JOBGRADE NOT NULL,
   job_country COUNTRYNAME NOT NULL,
   salary SALARY NOT NULL,
   full_name COMPUTED BY (last_name || ', ' || first_name),
   FOREIGN KEY (dept_no)
     REFERENCES Department (dept_no),
   FOREIGN KEY (job_code, job_grade, job_country)
     REFERENCES Job (job_code, job_grade, job_country),
   CHECK (
     salary >= (SELECT min_salary
                FROM job
                WHERE
                  Job.job_code = Employee.job_code AND
                  Job.job_grade = Employee.job_grade AND
                  Job.job_country = Employee.job_country)
     AND salary <= (SELECT max_salary
                    FROM Job
                    WHERE
                      Job.job_code = Employee.job_code AND
                      Job.job_grade = Employee.job_grade AND
                      Job.job_country = Employee.job_country) ) );

CREATE TABLE SALARY_HISTORY (
   emp_no EMPNO NOT NULL,
   change_date DATE DEFAULT 'NOW' NOT NULL,
   updater_id VARCHAR(20) NOT NULL,
   old_salary SALARY 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.

**Пример 11.15**

```
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 ;
```

### Триггеры, преобразующие неизменяемые представления в изменяемые

Одно из назначений триггеров — преобразование неизменяемых представлений в изменяемые. Примеры таких триггеров см. в главе 9 «Работа с представлениями», листинги 9.12 — 9.15.

## 11.6 Работа с хранимыми процедурами

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.

Существует два вида хранимых процедур — выполняемые хранимые процедуры (`executed stored procedures`) и хранимые процедуры выбора (`selected stored procedures`).

Выполняемые хранимые процедуры осуществляют обработку данных, находящихся в базе данных, или вовсе не связанных с базой данных. Эти процедуры могут получать входные параметры и возвращать выходные параметры. Обращение к выполняемым хранимым процедурам осуществляется при выполнении оператора SQL `EXECUTE PROCEDURE`.

Хранимые процедуры выбора как правило осуществляют выборку данных из базы данных, возвращая произвольное количество полученных строк. Процедуры выбора также могут получать входные параметры. Значение каждой очередной прочитанной строки возвращается вызвавшей программе в выходных параметрах. Для временной приостановки выполнения такой процедуры и передачи выбранных данных вызвавшей программе в хранимой процедуре используется оператор `SUSPEND`. Обращение к хранимой процедуре выбора осуществляется при помощи оператора `SELECT`.

### Создание хранимой процедуры

Синтаксически создание выполняемой хранимой процедуры от хранимой процедуры выбора никак не отличается. Для создания хранимой процедуры используется оператор `CREATE PROCEDURE`, синтаксис которого представлен в листинге 11.36.

**Листинг 11.36. 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
   [<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>
               | <объявление функции>
```

Хранимую процедуру может создать администратор и пользователь с привилегией CREATE PROCEDURE.

Имя хранимой процедуры может содержать до 31 символа и должно быть уникальным среди имен хранимых процедур базы данных, таблиц и представлений.

Хранимой процедуре от вызвавшей программы могут передаваться входные параметры. 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.

Хранимая процедура может возвращать вызвавшей программе произвольное количество выходных 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.

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.

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.

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 в следующих версиях Ред Базы Данных.

Используйте следующие предложения, чтобы процедура выполнялась:

- с правами вызывающего ее пользователя (значение по умолчанию)

```
SQL SECURITY INVOKER | AUTHID CALLER
```

- с правами ее владельца (создателя)

```
SQL SECURITY DEFINER | AUTHID OWNER
```

Значение по умолчанию на уровне всей базы данных можно изменить оператором ALTER DATABASE SET DEFAULT SQL SECURITY.

В теле хранимой процедуры может быть описано произвольное количество локальных переменных, именованных курсоров и подпрограммы (подпроцедуры и подфункции).

После описания локальных переменных в теле хранимой процедуры следует блок операторов, заключенных в операторные скобки BEGIN и END.

## Изменение хранимой процедуры

Для изменения существующей хранимой процедуры используется оператор ALTER PROCEDURE. Синтаксис оператора представлен в листинге 11.37.

**Листинг 11.37. Синтаксис оператора изменения хранимой процедуры** ALTER PROCEDURE

```
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
   [<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;

- тело хранимой процедуры.

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.

Изменять хранимую процедуру может ее создатель и администратор (пользователь с ролью RDB$ADMIN) и пользователь с привилегией ALTER ANY PROCEDURE.

## Создание новой или изменение существующей хранимой процедуры

Оператор CREATE OR ALTER PROCEDURE позволяет создать новую хранимую процедуру, если процедура с тем же именем отсутствует в базе данных, или изменить описание существующей в базе данных процедуры. Если процедура с этим именем уже существует, то происходит ее замена на новую хранимую процедуру, при этом существующие привилегии и зависимости сохраняются. Синтаксис оператора представлен в листинге 11.38.

**Листинг 11.38. 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
   [<объявление> [<объявление> ...] ]
  BEGIN
   <блок операторов>
  END }
```

Семантика операторов и предложений в этом операторе полностью соответствует оператору CREATE PROCEDURE.

## Удаление хранимой процедуры

Для удаления существующей хранимой процедуры используется оператор DROP PROCEDURE. Синтаксис оператора представлен в листинге 11.39.

**Листинг 11.39. Синтаксис оператора удаления хранимой процедуры 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.

## 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 11.40. 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
   [<объявление> [<объявление> ...] ]
  BEGIN
   <блок операторов>
  END }
```

Операция закончится неудачей при подтверждении транзакции, если процедура имеет зависимости.

## 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 написанная 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.

Хранимая процедура выбора 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.

### Получение значения искусственного первичного ключа

Для получения значения искусственного первичного ключа из генератора клиентской программой для таблицы PEOPLE можно использовать выполняемую хранимую процедуру:

**Пример 11.16**

```
SET TERM ^;
CREATE PROCEDURE PROC_PEOPLE
RETURNS (COD INTEGER)
AS
```

```
BEGIN
   COD = NEXT VALUE FOR GEN_PEOPLE;
END ^
```

Сравните этот текст с примером 11.33, 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.

## Вычисление факториала числа

В главе 8 «Операторы DML» в листинге 8.31 был приведен пример оператора `EXECUTE BLOCK`, который позволял вычислить факториал заданного числа в декларативной части SQL.

В следующем примере 11.17 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`.

**Пример 11.17**

```
CREATE OR ALTER PROCEDURE PROC_TEST8 (N BIGINT)
RETURNS (RESULT BIGINT, TEXT VARCHAR(50))
AS
   DECLARE VARIABLE I BIGINT;
BEGIN
   RESULT = 1; I = 1;
   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_TEST8 (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 примере 11.18 создается рекурсивная хранимая процедура, вычисляющая факториал. Пример взят из документации по InterBase.

**Пример 11.18**

```
SET TERM ^;
CREATE PROCEDURE FACTORIAL
   (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 FACTORIAL (NUM_LESS_ONE)
       RETURNING_VALUES N_FACTORIAL;
       N_FACTORIAL = N_FACTORIAL * NUM;
       SUSPEND;
     END
END ^
SET TERM ;^
```

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 примере 11.17 (как и в примере 11.40) выбран другой тип данных только для того, чтобы проиллюстрировать средства обработки ошибок в PSQL. Это касается и многих других примеров данной главы.

В примере 11.3 был приведен пример использования операторов PSQL для получения чисел Фибоначчи. В примере 11.19 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.

**Пример 11.19**

```
SET TERM ^;
CREATE PROCEDURE PROC_TEST1
   (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 хранимой процедуры выбора 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.

**Пример 11.20**

```
CREATE PROCEDURE PROC_TEST9
   (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;
   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 2 «Типы данных Ред База Данных».

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 PROC_TEST9(90);
```

Здесь для дробного числа также выполняется преобразование CAST для того, чтобы получить максимальное количество дробных знаков.

Далее приведен пример простой процедуры выбора, которая выбирает из таблицы REGION строки, принадлежащие одной стране. Код страны передается процедуре в качестве входного параметра.

**Пример 11.21**

```
CREATE PROCEDURE PROC_SELECT_REGION2
   (CODCOUNTRY CHAR(3))
RETURNS (CODREGION CHAR(2), NAMEREG CHAR(40), CENTER CHAR(25))
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
   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'
ORDER BY "Название региона";
```

В этом операторе выбираются только те регионы России, в названии которых присутствует буква «а». Результирующие строки упорядочиваются по названию региона.

# 11.7 Работа с хранимыми функциями

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.

## Создание хранимой функции

Для создания хранимой функции используется оператор `CREATE FUNCTION`, синтаксис которого представлен в листинге 11.41.

**Листинг 11.41. 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
   [<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>
               | <объявление функции>
```

Хранимую функцию может создать администратор и пользователь с привилегией `CREATE FUNCTION`. Пользователь, создавший хранимую функцию, становится её владельцем.

Имя хранимой функции должно быть уникальным среди имён всех хранимых функций и внешних (UDF) функций. Для внутренних функций достаточно уникальности только в рамках модулей, которые их «охватывают».

Хранимой функции от вызвавшей программы могут передаваться входные параметры. 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 строкового типа существует возможность задать порядок сортировки с помощью предложения COLLATE. Кроме того, для параметра можно указать ограничение NOT NULL, тем самым запретив передавать в него значение NULL.

Входные 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.

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).

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).

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.

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.

В теле хранимой функции может быть описано произвольное количество локальных переменных, именованных курсоров и подпрограммы (подпроцедуры и подфункции).

После описания локальных переменных в теле хранимой функции следует блок операторов, заключенных в операторные скобки BEGIN и END.

## Изменение хранимой функции

Для изменения существующей хранимой функции используется оператор ALTER FUNCTION. Синтаксис оператора представлен в листинге 11.42.

**Листинг 11.42. 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
```

```
   [<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;
- точку входа и имя движка (для внешних функций)

После выполнения существующие привилегии и зависимости сохраняются.

## 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 11.43.

**Листинг 11.43. 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
   [<объявление> [<объявление> ...] ]
  BEGIN
   <блок операторов>
  END }
```

## Удаление хранимой функции

Для удаления существующей хранимой функции используется оператор `DROP FUNCTION`. Синтаксис оператора представлен в листинге 11.44.

**Листинг 11.44. 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.

### 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 11.45.

**Листинг 11.45. 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
   [<объявление> [<объявление> ...] ]
  BEGIN
   <блок операторов>
  END }
```

Операция закончится неудачей при подтверждении транзакции, если функция имеет зависимости.

## 11.8 Работа с пакетами

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 (CREATE PACKAGE), а затем — тело (CREATE PACKAGE BODY).

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.

## Создание заголовка пакета

Оператор `CREATE PACKAGE` создаёт новый заголовок пакета. Синтаксис оператора представлен в листинге 11.46.

**Листинг 11.46. 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>
  | [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. Имена процедур и функций, объявленные в заголовке пакета, должны быть уникальны среди имён процедур и функций, объявленных в заголовке и теле пакета.

Подробное описание заголовков хранимых процедур и функций можно найти в соответствующих разделах (см. раздел 11.6 и раздел 11.7.)

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.

## Изменение заголовка пакета

Оператор ALTER PACKAGE изменяет заголовок пакета. Синтаксис оператора представлен в листинге 11.47.

**Листинг 11.47. 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. При этом исходный код тела пакета сохраняется. Состояние соответствия тела пакета его заголовку отображается в таблице RDB$PACKAGES в столбце RDB$VALID_BODY_FLAG. Также данным оператором можно поменять в контексте какого пользователя будет выполняться пакет.

## Создание нового или изменение существующего заголовка пакета

Оператор CREATE OR ALTER PACKAGE создаёт новый или изменяет существующий заголовок пакета. Синтаксис оператора представлен в листинге 11.48.

**Листинг 11.48. 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> ::=
   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.

## Удаление заголовка пакета

Оператор DROP PACKAGE удаляет существующий заголовок пакета. Синтаксис оператора представлен в листинге 11.49.

**Листинг 11.49. 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.

## Создание нового или пересоздание существующего заголовка объекта

Оператор RECREATE PACKAGE создаёт новый или пересоздает существующий заголовок пакета. Синтаксис оператора представлен в листинге 11.50.

**Листинг 11.50. 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> ::=
```

```
   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.

## 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 11.51.

**Листинг 11.51. 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
         [<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
         [<объявление> [<объявление> ...] ]
       BEGIN
         <блок операторов>
       END }

<входной параметр> ::= <описание параметра> [{=|DEFAULT} <значение по умолчанию>]
<входной_параметр> ::= <описание параметра>
<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>
                 | <объявление 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.

## Удаление тела пакета

Оператор DROP PACKAGE BODY удаляет существующее тело пакета. Синтаксис оператора представлен в листинге 11.52.

**Листинг 11.52. 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.

## Создание нового или пересоздание существующего тела объекта

Оператор RECREATE PACKAGE BODY создаёт новое или пересоздает существующее тело пакета. Синтаксис оператора представлен в листинге Д.81.

**Листинг 11.53. 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
         [<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
         [<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 12

# Внешние хранимые процедуры, функции и триггеры, написанные на языке Java

В «Ред База 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 База Данных» для работы с внешними подпрограммами описано в Руководстве Администратора. О взаимодействиях с Java методами из базы данных описано в Руководстве Разработчика. В этом же разделе рассмотрен SQL синтаксис операторов создания, изменения и пересоздания внешних хранимых подпрограмм, написанных на Java.

## 12.1 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 12.1. Синтаксис операторов создания, изменения и пересоздания внешней хранимой процедуры**

```
{CREATE [ OR ALTER ] | RECREATE | ALTER} PROCEDURE <имя хранимой процедуры>
   [(<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. FBJava передает каждый параметр как массив с длиной 1, таким образом процедуры могут менять их нулевой элемент.

### Примеры

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();}
}
```

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;
     }
   };
}
```

## 12.2 Объявление/изменение/пересоздание внешних функций

Синтаксис операторов создания, изменения и пересоздания внешней функции, написанной на Java, имеет одинаковую семантику и приведен в листинге:

**Листинг 12.2. Синтаксис операторов создания, изменения и пересоздания внешней функции**

```
{CREATE [ OR ALTER ] | RECREATE | ALTER} FUNCTION <имя функции>
   [(<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.

### Примеры

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;
```

2. Описанная в примере внешняя функция производит суммирование входных параметров функции, объявленной в базе данных:

```
public static int sum() {
   FunctionContext context = FunctionContext .get();
   ValuesMetadata valuesmetadata = context.getInputMetadata();
   Values values = context.getInputValues();
   int ret = 0;
   for (int i = valuesmetadata.getParameterCount(); i >= 1; ++i)
      ret += (Integer) values.getObject(i);
   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;
```

## 12.3 Объявление/изменение/пересоздание внешних триггеров

Синтаксис операторов создания, изменения и пересоздания внешнего триггера, написанного на Java, имеет одинаковую семантику и приведен в листинге:

**Листинг 12.3. Синтаксис операторов создания, изменения и пересоздания внешнего триггера**

```
{CREATE [ OR ALTER ] | RECREATE | 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 '<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.

### Примеры

1. Тело внешнего триггера, написанного на Java, выполняющего логгирование:

```
public static void info() throws SQLException {
   Logger log = LoggerFactory.getLogger(FbLogger.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);
   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;
```

## 12.4 Удаление внешних процедур, функций и триггеров

Удаление внешней процедуры осуществляется оператором `DROP PROCEDURE`.

**Листинг 12.4. 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 12.5. Синтаксис оператора удаления функции**

```
DROP FUNCTION <имя_функции>;
```

Оператор `DROP TRIGGER` удаляет существующий триггер:

**Листинг 12.6. Синтаксис оператора удаления триггера**

```
DROP TRIGGER <имя_триггера>;
```

## 12.5 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 13

# 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 3.0".

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 использует сторонние Java-библиотеки, которые могут содержать уязвимости безопасности. Пожалуйста, перед его настройкой изучите известные уязвимости используемых jar-файлов

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.

## 13.1 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 8.

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`.
2. В `plugins.conf` установите путь к JRE в `JavaHome`, например:

```
JavaHome = /usr/lib/jvm/java-8-openjdk-amd64/jre
```

3. Подключитесь к базе данных безопасности `java-security.fdb` и назначьте права доступа пользователям базы данных, использующим код Java (подробнее см. раздел Безопасность). В каталоге `misc` сервера находится файл `fts_permissions.sql` со скриптом по назначению прав доступа для работы `fbjava_lucene`. Он содержит базовый набор прав для работы с полнотекстовым поиском. Для назначения базовых прав необходимо выполнить скрипт `fts_permissions.sql` на базе `java-security.fdb`. Администратор может по своему усмотрению редактировать скрипт, менять название каталога для хранения индекса lucene или имя роли.

```
echo "input '/opt/RedDatabase/misc/fts_permissions.sql';" |
/opt/RedDatabase/bin/isql -u sysdba -p masterkey
localhost:/opt/RedDatabase/java-security.fdb
```

4. В файле конфигурации `fbjava.yaml` укажите пути к Java-библиотекам, которые реализуют функции полнотекстового поиска:

```
classpath:
   - $(root)/jar/fts/*.jar
```

5. Если для БД необходим отдельный каталог для хранения индексов, тогда в файле конфигурации `fbjava.yaml` добавьте в секцию `databases`, укажите БД в виде шаблона регулярного выражения, создайте секцию `options` с параметром `ftsDirectory`, в котором будет указан каталог:

```
databases:
    ".*/database.fdb":
       options:
          ftsDirectory:
             - /path/to/directory
```

6. В `jvm.args` установите директорию для хранения индекса (по умолчанию `/tmp/RDBLuceneIndex/`), если в `fbjava.yaml` для БД не указан отдельный каталог.

```
-Dfts.directory=...
```

7. По умолчанию в `fts.directory` для БД будет создан каталог (если в `fbjava.yaml` 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
```

8. 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), например:

```
-Dfts.max_document_size=10000
-Dfts.skip_corrupted=true
```

9. В `jvm.args` можно включить логирование из `FBJava`:

```
-Dfbjava.disable_output=false
```

Если необходимо логировать `JNA` вывод, то нужно указать:

```
-Djna.debug_load=true
-Djna.debug_load.jna=true
-Djna.dump_memory=true
```

Для логирования используется библиотека `log4j`. Файл `log4j.properties` задает свойства ведения журнала. По умолчанию файл настроен следующим образом:

```
log4j.rootLogger=ERROR,stdout
log4j.logger.com.endeca=INFO
log4j.logger.com.endeca.itl.web.metrics=INFO
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%p%dISO8601%r%c[%t]%m%n
```

Наличие только `ConsoleAppender` означает, что стандартный вывод будет направлен в консоль, а не в лог-файл.
Параметры логирования можно настроить с помощью следующей опции:

```
-Dlog4j.configuration=file://path/to/log4j.properties
```

Вы можете настроить `log4j.properties` таким образом, чтобы сообщения записывались только в файл или как в консоль, так и в файл. Например, вы могли бы изменить приведенную выше конфигурацию:

```
# инициализируйте корневой регистратор с уровнем ERROR для stdout и fout
log4j.rootLogger=ERROR,stdout,fout
# установите уровень логирования для этих компонентов
log4j.logger.com.endeca=INFO
log4j.logger.com.endeca.itl.web.metrics=INFO
# добавьте ConsoleAppender в stdout регистратора для записи в консоль
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
# используйте формат сообщения
log4j.appender.stdout.layout.ConversionPattern=%m%n
# добавьте FileAppender в регистратор fout
log4j.appender.fout=org.apache.log4j.FileAppender
# Создайте лог-файл
log4j.appender.fout.File=/path/to/crawl.log
log4j.appender.fout.layout=org.apache.log4j.PatternLayout
# используйте более подробный шаблон сообщения
log4j.appender.fout.layout.ConversionPattern=%p%dISO8601%r%c[%t]%m%n
```

В примере `FileAppender` записывает события в лог-файл с именем `crawl.log`, который создастся по указанному пути. У сервера должны быть права на запись для заданного каталога. Если путь к лог-файлу не указан, то по умолчанию он будет создан во временном каталоге

(\tmp или заданный системой). ConsoleAppender ведёт запись в консоль, используя шаблон, в котором отображаются только сообщения,но не более подробная информация.
Можно изменить уровни ведения журнала:

- DEBUG – обозначает детализированные информационные события, которые наиболее полезны для отладки конфигурации обхода.
- TRACE – обозначает более детализированные информационные события, чем DEBUG.
- ERROR – обозначает ошибки, с которыми можно продолжить работу.
- FATAL – означает серьезные ошибки, которые, могут привести к прерыванию работы.
- INFO – обозначает информационные сообщения, которые описывают процесс обхода на наиболее детальном уровне.
- OFF – имеет самый высокий из возможных возможный ранг и предназначен для отключения ведения журнала.
- WARN – обозначает потенциально опасные ситуации.

Эти уровни позволяют отслеживать интересующие события,с необходимой степенью детализации, не перегружаясь сообщениями, которые не имеют отношения к делу. Когда вы настраиваете конфигурацию обхода, вы можете использовать уровень DEBUG для получения всех сообщений и перейти на менее подробный уровень в процессе работы.
Обратите внимание, что файл log4j.properties по умолчанию содержит ряд предлагаемых регистраторов компонентов, которые закомментированы. Чтобы использовать любой из этих регистраторов, удалите символ комментария (#).

10. Перезапустите сервер.

## Безопасность

Одной из наиболее важных особенностей платформы Java является система безопасности, так называемая «песочница». FBJava 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 | Описание |
|---|---|---|
| PERMISSION_GROUP | ID, NAME | Название группы полномочий |
| PERMISSION | PERMISSION_GROUP, CLASS_NAME, ARG1, ARG2 | Права доступа Java, относящиеся к определенной группе полномочий |
| PERMISSION_GROUP_GRANT | PERMISSION_GROUP, DATABASE_PATTERN, GRANTEE_TYPE, GRANTEE_PATTERN | Кому (пользователю/роли) и для какой базы данных назначены права группы полномочий |

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.

Существует несколько предопределенных переменных, которые могут быть использованы в качестве значения аргумента ARG1, где требуется имя каталога. Полный их список выглядит следующим образом:

- $(root) – корневой каталог;
- $(install) – директория, куда установлена СУБД. Изначально $(root) и $(install) одинаковые. $(root) может быть переопределена установкой или изменением переменной

окружения FIREBIRD, в таком случае эта переменная отлична от $(install);

- $(jar) – путь до каталога jar корневой папки.

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.

При подключении пользователя кэшируются роли и права доступа, поэтому любые изменения политик и ролей вступят в силу только при следующем коннекте к базе данных.

База данных java-security.fdb изначально не пустая, в ней создана группа COMMON с некоторыми полномочиями для всех пользователей всех баз данных (шаблон %), а также группа RDB$ADMINS, предоставляющая все разрешения для системных администраторов. В группу RDB$ADMINS по умолчанию добавлен пользователь SYSDBA.

Добавление пользователей в группу RDB$ADMINS должно происходить с осторожностью, т.к. пользователи этой группы имеют разрешение на запуск любого кода.

В каталоге установки сервера можно найти файл misc/update-java-security.sql со скриптом по назначению полного доступа для группы системных администраторов, если база данных java-security.fdb изначально не включала права для них, в том числе, для пользователя SYSDBA.

Таблица 13.2 — 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 |

Таблица 13.3 — Предоставленные права для группы администраторов RDB$ADMINS

| CLASS_NAME | ARG1 | ARG2 |
|---|---|---|
| java.security.AllPermission | null | null |

**Пример**

Назначение прав доступа роли TESTUSER в любой базе данных.

```
insert into PERMISSION_GROUP values (100, 'TEST');
insert into PERMISSION_GROUP_GRANT values (100, '%', 'ROLE', 'TESTUSER');
insert into PERMISSION values (100, 'java.util.PropertyPermission', 'fg', 'read');
insert into PERMISSION values (100, 'java.util.PropertyPermission', 'java.home',
                                                                    'read');
insert into PERMISSION values (100, 'java.io.FilePermission', '$(jar)/fts/*',
```

```
'read')
```

# 13.2 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 корневом каталоге сервера.

```
isql -u sysdba -p masterkey -i /opt/RedDatabase/misc/fts.sql
localhost:/tmp/test_fts.fdb
```

## Служебные домены

В полнотекстовом поиске используются три служебных домена.

Таблица 13.4 — 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. |

## Служебные таблицы

В полнотекстовом поиске используются три служебные таблицы.

### FTS$INDICES

В служебной таблице FTS$INDICES хранятся метаданные индексов.

Таблица 13.5 — 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 |
| FTS$ANALYZER | VARCHAR(255) CHARACTER SET UNICODE_FSS | Служебная информация, имя анализатора |

Таблица 13.5 — Структура таблицы FTS$INDICES

| Имя поля | Тип | Описание |
|---|---|---|
| FTS$INDEX_STATUS | CHAR(1) CHARACTER SET UNICODE_FSS | Статус индекса. Это поле может принимать следующие значения:<br>• 'I' inactive – индекс неактивный;<br>• 'N' new – индекс создан, требует полное переиндексирование;<br>• 'U' needs metadata update – требуется изменение метаданных, триггеров и т.д.;<br>• 'D' drop – индекс отмечен к удалению;<br>• 'C' complete – для индекса сделаны все изменения в метаданных и он индексируется |
| FTS$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 13.6 — 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$FIELD_NAME | VARCHAR(31) CHARACTER SET UNICODE_FSS | Индексируемое поле |
| FTS$TRIGGER_NAME | VARCHAR(31) CHARACTER SET UNICODE_FSS | Имя триггера, который будет заполнять таблицу FTS$POOL после изменения данных |

## FTS$POOL

Таблица FTS$POOL содержит значения RDB$DB_KEY[3] для измененных, но не проиндексированных полей.

[3] 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).

Таблица 13.7 — Структура таблицы FTS$POOL

| Имя поля | Тип | Описание |
|---|---|---|
| FTS$ROW_ID | CHAR(8) CHARACTER SET OCTETS | RDB$DB_KEY записи, которая была добавлена, изменена или удалена |
| FTS$STATUS | SMALLINT | Отображает статус записи. Может принимать 3 значения:<br>• 1 - запись добавлена;<br>• 2 - запись изменена;<br>• 3 - запись удалена. |

# Служебные хранимые процедуры

## FTS$CREATE_INDEX

Процедура FTS$CREATE_INDEX используется для создания индекса.

Таблица 13.8 — 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.[4] |
| 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 13.9 — Соответствие имен анализаторов и языков

| Имя анализатора | Язык |
|---|---|
| English | Английский |
| Standard | Английский |
| Russian | Русский |
| German | Немецкий |
| French | Французский |
| Czech | Чешский |
| Brazilian | Бразильский |

[4] Допустимые значения имен анализаторов приведены в таблице 13.9.

Таблица 13.9 — Соответствие имен анализаторов и языков

| Имя анализатора | Язык |
|---|---|
| Chinese | Китайский |
| Dutch | Голландский |
| 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 13.10 — Входные параметры процедуры FTS$DROP_INDEX

| Имя поля | 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 13.11 — Входные параметры процедуры FTS$ADD_FIELD_TO_INDEX

| 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$DROP_FIELD_FROM_INDEX

Процедура FTS$DROP_FIELD_FROM_INDEX используется для удаления индексируемого поля из индекса.

Таблица 13.12 — Входные параметры процедуры FTS$DROP_FIELD_FROM_INDEX

| 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$REINDEX

Служебная процедура FTS$REINDEX производит полную (по всем записям, с заменой ранее существующего индекса) переиндексацию указанного индекса.

Таблица 13.13 — Входные параметры процедуры FTS$REINDEX

| Имя поля | 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 13.14 — 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. http://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 13.15 — 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$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[5] 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.

## 13.3 Пример использования полнотекстового поиска

Допустим в базе данных создана таблица TEST_TABLE:

```
create table TEST_TABLE(id int, 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).

### 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 13.9. По умолчанию используется значение Standard. Значение третьего параметра задает описание для индекса.

```
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.

[5] Демон – это процесс, работающий в фоновом режиме без прямого общения с пользователем.

## 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.

## 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.

## 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;
```

## 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. |

## 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.

# 13.4 Синтаксис поисковых запросов

## 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.

## 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 «`*`».

## 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"`.

## 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"
```

## 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. 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`. 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")
```

## 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: http://lucene.apache.org/.

# 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

| | | |
|---|---|---|
| !< | != | !> |
| ( | ) | , |
| < | <= | <> |
| = | > | >= |
| := | ^< | ^= |
| ^> | \|\| | ~< |
| ~= | ~> | ADAPTER |
| ADD | ADMIN | ALL |
| ALTER | AND | ANY |
| AS | AT | AVG |
| BEGIN | BETWEEN | BIGINT |
| BIT_LENGTH | BLOB | BOOLEAN |
| BOTH | BY | CASE |
| CAST | CHAR | CHARACTER |
| CHARACTER_LENGTH | CHAR_LENGTH | CHECK |
| CLOSE | COLLATE | COLUMN |
| COMMIT | CONNECT | CONSTRAINT |
| CORR | COUNT | COVAR_POP |
| COVAR_SAMP | CREATE | CROSS |
| CURRENT | CURRENT_CONNECTION | CURRENT_DATE |
| CURRENT_ROLE | CURRENT_TIME | CURRENT_TIMESTAMP |
| CURRENT_TRANSACTION | CURRENT_USER | CURSOR |
| DATE | DAY | DEC |
| 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 |
| INTEGER | INTO | IS |
| JOIN | LEADING | LEFT |
| LIKE | 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 | RDB$DB_KEY |
| RDB$RECORD_VERSION | REAL | RECORD_VERSION |
| RECREATE | RECURSIVE | REFERENCES |
| REGR_AVGX | REGR_AVGY | REGR_COUNT |
| REGR_INTERCEPT | REGR_R2 | REGR_SLOPE |
| REGR_SXX | REGR_SXY | REGR_SYY |
| RELEASE | 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 | TO |
| TRAILING | TRIGGER | TRIM |
| TRUE | UNION | UNIQUE |
| UNKNOWN | UPDATE | UPDATING |
| UPPER | USER | USING |

| | | |
|---|---|---|
| VALUE | VALUES | VARCHAR |
| VARIABLE | VARYING | VAR_POP |
| VAR_SAMP | VIEW | WHEN |
| WHERE | WHILE | WITH |
| 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 | AFTER | ALWAYS |
| ASC | ASCENDING | ASCII_CHAR |
| ASCII_VAL | ASIN | ASINH |
| ATAN | ATAN2 | ATANH |
| AUTO | AUTONOMOUS | BACKUP |
| BEFORE | BIN_AND | BIN_NOT |
| BIN_OR | BIN_SHL | BIN_SHR |
| BIN_XOR | BLOCK | BODY |
| BREAK | CALLER | CASCADE |
| CEIL | CEILING | CHAR_TO_UUID |
| COALESCE | COLLATION | COMMENT |
| COMMITTED | COMMON | COMPUTED |
| CONDITIONAL | CONTAINING | CONTINUE |
| COS | COSH | COT |
| CPU_LOAD | CREATE_FILE | CSTRING |
| CURRENT_LABEL | DAMLEV | DATA |
| DATABASE | DATEADD | DATEDIFF |
| DDL | DECODE | DECRYPT |
| DEFINER | DELETE_FILE | DENSE_RANK |
| DESC | DESCENDING | DESCRIPTOR |
| DIFFERENCE | DO | DOMAIN |
| ENCRYPT | ENGINE | ENTRY_POINT |
| EXCEPTION | EXIT | EXP |
| FILE | FIRST | FIRSTNAME |
| FIRST_VALUE | FLOOR | FREE_IT |
| GENERATED | GENERATOR | GEN_ID |
| GEN_UUID | GRANTED | HASH |
| HASH_CP | IDENTITY | IF |

| | | |
|---|---|---|
| IGNORE | IIF | INACTIVE |
| INCREMENT | INITIAL_LABEL | INPUT_TYPE |
| INVOKER | ISOLATION | IS_LABEL_VALID |
| KEY | LAG | LAST |
| LASTNAME | LAST_VALUE | LDAP_ATTR |
| LDAP_GROUPS | | |
| LDAP_USER_GROUPS | | |
| LDAP_ADMIN | | |
| LEAD | LEAVE | LENGTH |
| LEVEL | LIMBO | LINGER |
| LIST | LN | LOCK |
| LOG | LOG10 | LPAD |
| MANUAL | MAPPING | MATCHED |
| MATCHING | MAXVALUE | MIDDLENAME |
| MILLISECOND | MINVALUE | MOD |
| MODULE_NAME | NAME | NAMES |
| NEXT | NTH_VALUE | NULLIF |
| NULLS | OPTIMIZE | OPTION |
| OS_NAME | OUTPUT_TYPE | OVERFLOW |
| OVERLAY | PACKAGE | PAD |
| PAGE | PAGES | PAGE_SIZE |
| PARTITION | PASSWORD | PI |
| PLACING | PLUGIN | POWER |
| PRESERVE | PRIOR | PRIVILEGES |
| PROTECTED | RAND | RANK |
| RDB$GET_CONTEXT | RDB$ROLE_IN_USE | RDB$SET_CONTEXT |
| READ | READ_FILE | REGEXP_SUBSTR |
| RELATIVE | REPLACE | REQUESTS |
| RESERV | RESERVING | RESTART |
| RESTRICT | RETAIN | RETURNING |
| REVERSE | ROLE | ROUND |
| ROW_NUMBER | RPAD | SCALAR_ARRAY |
| SCHEMA | SECURITY | SEGMENT |
| SEQUENCE | SERVERWIDE | SHADOW |
| SHARED | SIGN | SIN |
| SINGULAR | SINH | SIZE |
| SKIP | SNAPSHOT | SORT |
| SOURCE | SPACE | SQL |
| SQRT | STABILITY | STARTING |

| | | |
|---|---|---|
| STARTS | STATEMENT | STATISTICS |
| SUBSTRING | SUB_TYPE | SUSPEND |
| TAGS | TAN | TANH |
| TEMPORARY | TIMEOUT | TRANSACTION |
| TRUNC | TRUSTED | TWO_PHASE |
| TYPE | UNCOMMITTED | UNDO |
| USAGE | UTC_TIMESTAMP | UUID_TO_CHAR |
| WAIT | WEEK | WEEKDAY |
| WORK | WRITE | YEARDAY |

# 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<br>Арифметическое исключение, числовое переполнение или строковое обрезание. |
| -901 | 335544322 | bad_dbkey | invalid database key<br>Неверный ключ базы данных. |
| -922 | 335544323 | bad_db_format | file @1 is not a valid database<br>Файл @1 не является допустимой базой данных |
| -904 | 335544324 | bad_db_handle | invalid database handle (no active connection)<br>Неверный дескриптор базы данных (нет активного соединения) |
| -924 | 335544325 | bad_dpb_content | bad parameters on attach or create database<br>Неверные параметры при подключении или при создании базы данных. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -901 | 335544326 | bad_dpb_form | unrecognized database parameter block<br>Блок параметров базы данных не распознан. |
| -901 | 335544327 | bad_req_handle | invalid request handle<br>Неверный запрос дескриптора. |
| -901 | 335544328 | bad_segstr_handle | invalid BLOB handle<br>Неверный дескриптор BLOB. |
| -901 | 335544329 | bad_segstr_id | invalid BLOB ID<br>Неверный идентификатор BLOB. |
| -901 | 335544330 | bad_tpb_content | invalid parameter in transaction parameter block<br>Неверный параметр в блоке параметров транзакции. |
| -901 | 335544331 | bad_tpb_form | invalid format for transaction parameter block<br>Неверный формат блока параметров транзакции. |
| -901 | 335544332 | bad_trans_handle | invalid transaction handle (expecting explicit transaction start)<br>Неверный дескриптор транзакции (ожидается явный запуск транзакции). |
| -902 | 335544333 | bug_check | internal Firebird consistency check (@1)<br>Внутренняя проверка целостности программного обеспечения (@1) |
| -413 | 335544334 | convert_error | conversion error from string "@1"<br>Ошибка преобразования для строки "@1". |
| -902 | 335544335 | db_corrupt | database file appears corrupt (@1)<br>Файл базы данных является поврежденным (@1) |
| -913 | 335544336 | deadlock | deadlock<br>Взаимная блокировка. |
| -901 | 335544337 | excess_trans | attempt to start more than @1 transactions<br>Не используется. |
| 100 | 335544338 | from_no_match | no match for first value expression<br>Нет соответствия для первого значения выражения. |
| -901 | 335544339 | infinap | information type inappropriate for object specified<br>Информационный тип не соответствует указанному объекту. |
| -901 | 335544340 | infona | no information of this type available for object specified<br>Не используется. |
| -901 | 335544341 | infunk | unknown information item<br>Неизвестный информационный элемент. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -901 | 335544342 | integ_fail | action cancelled by trigger (@1) to preserve data integrity<br>Действие отменено триггером (@1), чтобы сохранить целостность данных. |
| -104 | 335544343 | invalid_blr | invalid request BLR at offset @1<br>Неверный запрос BLR со смещением @1. |
| -902 | 335544344 | io_error | I/O error during "@1" operation for file "@2"<br>Ошибка ввода - вывода во время операции @1 для файла @2 |
| -901 | 335544345 | lock_conflict | lock conflict on no wait transaction<br>Конфликт блокировки для транзакции NO WAIT |
| -902 | 335544346 | metadata_corrupt | corrupt system table<br>Повреждение системной таблицы |
| -625 | 335544347 | not_valid | validation error for column @1, value "@2"<br>Ошибка проверки данных для столбца @1, значение "@2". |
| -508 | 335544348 | no_cur_rec | no current record for fetch operation<br>Нет текущей записи для операции FETCH. |
| -803 | 335544349 | no_dup | attempt to store duplicate value (visible to active transactions) in unique index "@1"<br>Попытка сохранить дубликат значения (видимые при активных транзакциях) в уникальном индексе "@1". |
| -901 | 335544350 | no_finish | program attempted to exit without finishing database<br>Не используется. |
| -607 | 335544351 | no_meta_update | unsuccessful metadata update<br>Неудачное обновление метаданных. |
| -551 | 335544352 | no_priv | no permission for @1 access to @2 @3<br>Не существует полномочий для доступа @1 к @2 @3. |
| -901 | 335544353 | no_recon | transaction is not in limbo<br>Транзакция не является зависшей (limbo). |
| 100 | 335544354 | no_record | invalid database key<br>Не используется. |
| -901 | 335544355 | no_segstr_close | BLOB was not closed<br>Не используется. |
| -820 | 335544356 | obsolete_metadata | metadata is obsolete<br>Не используется. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -901 | 335544357 | open_trans | cannot disconnect database with open transactions (@1 active)<br>Невозможно отключиться от базы данных при наличии открытой транзакции (активная транзакция @1). |
| -901 | 335544358 | port_len | message length error (encountered @1, expected @2)<br>Ошибка длины сообщения (встречено @1, ожидается @2) |
| -151 | 335544359 | read_only_field | attempted update of read-only column<br>Попытка обновить доступный только для чтения столбец. |
| -150 | 335544360 | read_only_rel | attempted update of read-only table<br>Не используется. |
| -817 | 335544361 | read_only_trans | attempted update during read-only transaction<br>Попытка выполнить изменения во время выполнения транзакции только для чтения. |
| -150 | 335544362 | read_only_view | cannot update read-only view @1<br>Невозможно изменить представление @1 только для чтения. |
| -901 | 335544363 | req_no_trans | no transaction for request<br>Для запроса нет транзакции. |
| -901 | 335544364 | req_sync | request synchronization error<br>Ошибка синхронизации запроса. |
| -901 | 335544365 | req_wrong_db | request referenced an unavailable database<br>Не используется. |
| 101 | 335544366 | segment | segment buffer length shorter than expected<br>Длина сегмента буфера меньше, чем ожидается. |
| 100 | 335544367 | segstr_eof | attempted retrieval of more segments than exist<br>Попытка обращения к сегменту большему, чем их существует |
| -402 | 335544368 | segstr_no_op | attempted invalid operation on a BLOB<br>Не используется. |
| -901 | 335544369 | segstr_no_read | attempted read of a new, open BLOB<br>Не используется. |
| -901 | 335544370 | segstr_no_trans | attempted action on BLOB outside transaction<br>Не используется. |
| -817 | 335544371 | segstr_no_write | attempted write to read-only BLOB<br>Попытка записи в тип данных BLOB только для чтения. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -901 | 335544372 | segstr_wrong_db | attempted reference to BLOB in unavailable database<br>Не используется. |
| -902 | 335544373 | sys_request | operating system directive @1 failed<br>Директива операционной системы @1 не удалась |
| -596 | 335544374 | stream_eof | attempt to fetch past the last record in a record stream<br>Попытка получения в потоке записей записи, следующей за последней. |
| -904 | 335544375 | unavailable | unavailable database<br>Недоступная база данных |
| -901 | 335544376 | unres_rel | table @1 was omitted from the transaction reserving list<br>Не используется. |
| -901 | 335544377 | uns_ext | request includes a DSRI extension not supported in this implementation<br>Запрос включает расширение DSRI, не поддерживаемое в этой реализации. |
| -901 | 335544378 | wish_list | feature is not supported<br>Возможность не поддерживается. |
| -820 | 335544379 | wrong_ods | unsupported on-disk structure for file @1; found @2.@3, support @4.@5<br>Неподдерживаемая ODS для файла @1, обнаружена @2.@3, поддерживается @4.@5. |
| -804 | 335544380 | wronumarg | wrong number of arguments on call<br>Не используется. |
| -904 | 335544381 | imp_exc | Implementation limit exceeded<br>Исчерпан лимит выполнения |
| -901 | 335544382 | random | @1 |
| -901 | 335544383 | fatal_conflict | unrecoverable conflict with limbo transaction @1<br>Не используется. |
| -902 | 335544384 | badblk | internal error<br>Внутренняя ошибка. |
| -902 | 335544385 | invpoolcl | internal error<br>Не используется. |
| -904 | 335544386 | nopoolids | too many requests<br>Не используется. |
| -902 | 335544387 | relbadblk | internal error<br>Не используется. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -902 | 335544388 | blktoobig | block size exceeds implementation restriction<br>Размер блока превышает ограничение реализации |
| -904 | 335544389 | bufexh | buffer exhausted<br>Не используется. |
| -104 | 335544390 | syntaxerr | BLR syntax error: expected @1 at offset @2, encountered @3<br>Ошибка синтаксиса BLR: ожидается @1 по смещению @2, встречено @3. |
| -904 | 335544391 | bufinuse | buffer in use<br>Не используется. |
| -901 | 335544392 | bdbincon | internal error<br>Не используется. |
| -904 | 335544393 | reqinuse | request in use<br>Запрос используется. |
| -902 | 335544394 | badodsver | incompatible version of on-disk structure<br>Не используется. |
| -219 | 335544395 | relnotdef | table @1 is not defined<br>Таблица @1 не определена. |
| -205 | 335544396 | fldnotdef | column @1 is not defined in table @2<br>Столбец @1 не определен в таблице @2. |
| -902 | 335544397 | dirtypage | internal error<br>Не используется. |
| -902 | 335544398 | waifortra | internal error<br>Не используется. |
| -902 | 335544399 | doubleloc | internal error<br>Не используется. |
| -902 | 335544400 | nodnotfnd | internal error<br>Не используется. |
| -902 | 335544401 | dupnodfnd | internal error<br>Не используется. |
| -902 | 335544402 | locnotmar | internal error<br>Не используется. |
| -689 | 335544403 | badpagtyp | page @1 is of wrong type (expected @2, found @3)<br>Страница @1 имеет неверный тип (ожидается @2, обнаружена @3). |
| -902 | 335544404 | corrupt | database corrupted<br>База данных повреждена. |
| -902 | 335544405 | badpage | checksum error on database page @1<br>Не используется. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -902 | 335544406 | badindex | index is broken<br>Не используется. |
| -901 | 335544407 | dbbnotzer | database handle not zero<br>Не используется. |
| -901 | 335544408 | tranotzer | transaction handle not zero<br>Не используется. |
| -902 | 335544409 | trareqmis | transaction-request mismatch (synchronization error)<br>Транзакция - несогласованный запрос (ошибка синхронизации) |
| -902 | 335544410 | badhndcnt | bad handle count<br>Не используется. |
| -902 | 335544411 | wrotpbver | wrong version of transaction parameter block<br>Неверная версия блока параметров транзакции |
| -902 | 335544412 | wroblrver | unsupported BLR version (expected @1, encountered @2)<br>Не поддерживаемая версия BLR ( ожидается @1 встретилась @2) |
| -902 | 335544413 | wrodpbver | wrong version of database parameter block<br>Неверная версия блока параметров базы данных. |
| -402 | 335544414 | blobnotsup | BLOB and array data types are not supported for @1 operation<br>Типы данных BLOB и массив не поддерживаются для операции @1. |
| -902 | 335544415 | badrelation | database corrupted<br>Не используется. |
| -902 | 335544416 | nodetach | internal error<br>Не используется. |
| -902 | 335544417 | notremote | internal error<br>Не используется. |
| -901 | 335544418 | trainlim | transaction in limbo<br>Зависшая транзакция. |
| -901 | 335544419 | notinlim | transaction not in limbo<br>Не используется. |
| -901 | 335544420 | traoutsta | transaction outstanding<br>Не используется. |
| -923 | 335544421 | connect_reject | connection rejected by remote interface<br>Соединение отменено удаленным интерфейсом |
| -902 | 335544422 | dbfile | internal error<br>Внутренняя ошибка. |
| -902 | 335544423 | orphan | internal error<br>Внутренняя ошибка. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -904 | 335544424 | no_lock_mgr | no lock manager available<br>Не используется. |
| -104 | 335544425 | ctxinuse | context already in use (BLR error)<br>Контекст находится в использовании (ошибка BLR ). |
| -104 | 335544426 | ctxnotdef | context not defined (BLR error)<br>Контекст не определен (ошибка BLR). |
| -402 | 335544427 | datnotsup | data operation not supported<br>Операция данных не поддерживается. |
| -901 | 335544428 | badmsgnum | undefined message number<br>Неопределенный номер сообщения |
| -104 | 335544429 | badparnum | undefined parameter number<br>Неверный номер параметра. |
| -904 | 335544430 | virmemexh | unable to allocate memory from operating system<br>Невозможно выделить память в операционной системе. |
| -901 | 335544431 | blocking_signal | blocking signal has been received<br>Не используется. |
| -902 | 335544432 | lockmanerr | lock manager error<br>Ошибка менеджера блокировок. |
| -924 | 335544433 | journerr | communication error with journal "@1"<br>Не используется. |
| -664 | 335544434 | keytoobig | key size exceeds implementation restriction for index "@1"<br>Размер ключа превышает ограничения реализации для индекса "@1" |
| -407 | 335544435 | nullsegkey | null segment of UNIQUE KEY<br>Не используется. |
| -902 | 335544436 | sqlerr | SQL error code = @1<br>Код SQL ошибки = @1 |
| -820 | 335544437 | wrodynver | wrong DYN version<br>Не используется. |
| -172 | 335544438 | funnotdef | function @1 is not defined<br>Функция @1 не определена. |
| -171 | 335544439 | funmismat | function @1 could not be matched<br>Функции @1 нельзя найти соответствие. |
| -104 | 335544440 | bad_msg_vec | Не используется. |
| -924 | 335544441 | bad_detach | database detach completed with errors<br>Не используется. |
| -901 | 335544442 | noargacc_read | database system cannot read argument @1<br>Не используется. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -901 | 335544443 | noargacc_write | database system cannot write argument @1<br>Не используется. |
| -817 | 335544444 | read_only | operation not supported<br>Операция не поддерживается. |
| -677 | 335544445 | ext_err | @1 extension error<br>Не используется. |
| -150 | 335544446 | non_updatable | not updatable<br>Не используется. |
| -926 | 335544447 | no_rollback | no rollback performed<br>Не используется. |
| -902 | 335544448 | bad_sec_info | Не используется. |
| -902 | 335544449 | invalid_sec_info | Не используется. |
| -901 | 335544450 | misc_interpreted | @1 |
| -904 | 335544451 | update_conflict | update conflicts with concurrent update<br>Изменение конфликтует с конкурирующим изменением. |
| -906 | 335544452 | unlicensed | product @1 is not licensed<br>Продукт @ 1 не лицензирован. |
| -904 | 335544453 | obj_in_use | object @1 is in use<br>Объект @1 используется. |
| -413 | 335544454 | nofilter | filter not found to convert type @1 to type @2<br>Не найден фильтр для преобразования типа @1 в тип @2 (для BLOB). |
| -904 | 335544455 | shadow_accessed | cannot attach active shadow file<br>Невозможно соединиться с активным файлом теневой копии |
| -104 | 335544456 | invalid_sdl | invalid slice description language at offset @1<br>Неверный фрагмент языка описания по смещению @1. |
| -406 | 335544457 | out_of_bounds | subscript out of bounds<br>Выход за пределы диапазона. |
| -171 | 335544458 | invalid_dimension | column not array or invalid dimensions (expected @1, encountered @2)<br>Столбец не является массивом или неверная размерность(ожидается @1, встретилась @2). |
| -911 | 335544459 | rec_in_limbo | record from transaction @1 is stuck in limbo<br>Запись транзакции @1 становится зависшей |
| -904 | 335544460 | shadow_missing | a file in manual shadow @1 is unavailable<br>Файл в ручной теневой копии @1 недоступен |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -923 | 335544461 | cant_validate | secondary server attachments cannot validate databases<br>Вторичные подключения к серверу не могут проверять базы данных. |
| -923 | 335544462 | cant_start_journal | secondary server attachments cannot start journaling<br>Вторичные подключения к серверу не могут начать журналирование. |
| -204 | 335544463 | gennotdef | generator @1 is not defined<br>Генератор @1 не определен. |
| -923 | 335544464 | cant_start_logging | secondary server attachments cannot start logging<br>Не используется. |
| -685 | 335544465 | bad_segstr_type | invalid BLOB type for operation<br>Неверный тип BLOB для операции. |
| -530 | 335544466 | foreign_key | violation of FOREIGN KEY constraint "@1" on table "@2"<br>Нарушение ограничения внешнего ключа @1 для таблицы @2. |
| -820 | 335544467 | high_minor | minor version too high found @1 expected @2<br>Не используется. |
| -901 | 335544468 | tra_state | transaction @1 is @2<br>Транзакция @1 является @2 |
| -532 | 335544469 | trans_invalid | transaction marked invalid and cannot be committed<br>Транзакция помечена как недействительная из-за ошибки ввода-вывода. |
| -902 | 335544470 | buf_invalid | cache buffer for page @1 invalid<br>Неверный буфер кэша для страницы @1. |
| -902 | 335544471 | indexnotdefined | there is no index in table @1 with id @2<br>Не используется. |
| -902 | 335544472 | login | Your user name and password are not defined. Ask your database administrator to set up a Firebird login.<br>Не определены ваше имя и пароль пользователя. Чтобы установить соединение с Firebird обратитесь к администратору базы данных. |
| -823 | 335544473 | invalid_bookmark | invalid bookmark handle<br>Не используется. |
| -824 | 335544474 | bad_lock_level | invalid lock level @1<br>Не используется. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -615 | 335544475 | relation_lock | lock on table @1 conflicts with existing lock<br>Блокировка в таблице @1 конфликтует с существующей блокировкой. |
| -615 | 335544476 | record_lock | requested record lock conflicts with existing lock<br>Запрашиваемая блокировка записи конфликтует с уже существующей блокировкой. |
| -692 | 335544477 | max_idx | maximum indexes per table (@1) exceeded<br>Превышено максимальное количество индексов для таблицы (@1). |
| -902 | 335544478 | jrn_enable | enable journal for database before starting online dump<br>Не используется. |
| -902 | 335544479 | old_failure | online dump failure. Retry dump<br>Не используется. |
| -902 | 335544480 | old_in_progress | an online dump is already in progress<br>Не используется. |
| -902 | 335544481 | old_no_space | no more disk/tape space. Cannot continue online dump<br>Не используется. |
| -902 | 335544482 | no_wal_no_jrn | journaling allowed only if database has Write-ahead Log<br>Не используется. |
| -902 | 335544483 | num_old_files | maximum number of online dump files that can be specified is 16<br>Не используется. |
| -902 | 335544484 | wal_file_open | error in opening Write-ahead Log file during recovery<br>Не используется. |
| -901 | 335544485 | bad_stmt_handle | invalid statement handle<br>Неверный дескриптор оператора. |
| -902 | 335544486 | wal_failure | Write-ahead log subsystem failure<br>Не используется. |
| -230 | 335544487 | walw_err | WAL Writer error<br>Не используется. |
| -231 | 335544488 | logh_small | Log file header of @1 too small<br>Не используется. |
| -232 | 335544489 | logh_inv_version | Invalid version of log file @1<br>Не используется. |
| -233 | 335544490 | logh_open_flag | Log file @1 not latest in the chain but open flag still set<br>Не используется. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -234 | 335544491 | logh_open_flag2 | Log file @1 not closed properly; database recovery may be required<br>Не используется. |
| -235 | 335544492 | logh_diff_dbname | Database name in the log file @1 is different<br>Не используется. |
| -236 | 335544493 | logf_unexpected_eof | Unexpected end of log file @1 at offset @2<br>Не используется. |
| -237 | 335544494 | logr_incomplete | Incomplete log record at offset @1 in log file @2<br>Не используется. |
| -238 | 335544495 | logr_header_small | Log record header too small at offset @1 in log file @2<br>Не используется. |
| -239 | 335544496 | logb_small | Log block too small at offset @1 in log file @2<br>Не используется. |
| -240 | 335544497 | wal_illegal_attach | Illegal attempt to attach to an uninitialized WAL segment for @1<br>Не используется. |
| -241 | 335544498 | wal_invalid_wpb | Invalid WAL parameter block option @1<br>Не используется. |
| -242 | 335544499 | wal_err_rollover | Cannot roll over to the next log file @1<br>Не используется. |
| -243 | 335544500 | no_wal | database does not use Write-ahead Log<br>База данных не использует запись с упреждением лога. |
| -615 | 335544501 | drop_wal | cannot drop log file when journaling is enabled<br>Невозможно удалить лог файл, пока включено журналирование. |
| -204 | 335544502 | stream_not_defined | reference to invalid stream number<br>Ссылка на неверный номер потока. |
| -244 | 335544503 | wal_subsys_error | WAL subsystem encountered error<br>Не используется. |
| -245 | 335544504 | wal_subsys_corrupt | WAL subsystem corrupted<br>Не используется. |
| -902 | 335544505 | no_archive | must specify archive file when enabling long term journal for databases with round-robin log files<br>Не используется. |
| -902 | 335544506 | shutinprog | database @1 shutdown in progress<br>Выполняется останов базы данных @1 |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -615 | 335544507 | range_in_use | refresh range number @1 already in use<br>Не используется. |
| -834 | 335544508 | range_not_found | refresh range number @1 not found<br>Не используется. |
| -204 | 335544509 | charset_not_found | CHARACTER SET @1 is not defined<br>Набор символов @1 не определен. |
| -901 | 335544510 | lock_timeout | lock time-out on wait transaction<br>Истечение времени ожидания блокировки для транзакции WAIT. |
| -204 | 335544511 | prcnotdef | procedure @1 is not defined<br>Процедура @1 не определена. |
| -170 | 335544512 | prcmismat | Input parameter mismatch for procedure @1<br>Входные параметры не соответствуют для процедуры @1. |
| -246 | 335544513 | wal_bugcheck | Database @1: WAL subsystem bug for pid @23<br>Не используется. |
| -247 | 335544514 | wal_cant_expand | Could not expand the WAL segment for database @1<br>Не используется. |
| -204 | 335544515 | codnotdef | status code @1 unknown<br>Неизвестный код состояния @1. |
| -204 | 335544516 | xcpnotdef | exception @1 not defined<br>Не определено исключение @1. |
| -836 | 335544517 | except | exception @1<br>Исключение @1. |
| -837 | 335544518 | cache_restart | restart shared cache manager<br>Повторный запуск менеджера совместно используемого кэша. |
| -825 | 335544519 | bad_lock_handle | invalid lock handle<br>Не используется. |
| -902 | 335544520 | jrn_present | long-term journaling already enabled<br>Не используется. |
| -248 | 335544521 | wal_err_rollover2 | Unable to roll over please see Firebird log.<br>Не используется. |
| -249 | 335544522 | wal_err_logwrite | WAL I/O error. Please see Firebird log.<br>Не используется. |
| -250 | 335544523 | wal_err_jrn_comm | WAL writer - Journal server communication error. Please see Firebird log.<br>Не используется. |
| -251 | 335544524 | wal_err_expansion | WAL buffers cannot be increased. Please see Firebird log.<br>Не используется. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -252 | 335544525 | wal_err_setup | WAL setup error. Please see Firebird log.<br>Не используется. |
| -253 | 335544526 | wal_err_ww_sync | obsolete<br>Не используется. |
| -254 | 335544527 | wal_err_ww_start | Cannot start WAL writer for the database @1<br>Не используется. |
| -902 | 335544528 | shutdown | database @1 shutdown<br>База данных @1 остановлена. |
| -553 | 335544529 | existing_priv_mod | cannot modify an existing user privilege<br>Невозможно изменить существующую привилегию пользователя. |
| -616 | 335544530 | primary_key_ref | Cannot delete PRIMARY KEY being used in FOREIGN KEY definition.<br>Невозможно удалить первичный ключ, который используется в определении внешнего ключа. |
| -291 | 335544531 | primary_key_notnull | Column used in a PRIMARY constraint must be NOT NULL.<br>Столбец, используемый в ограничении первичного ключа, должен быть NOT NULL. |
| -204 | 335544532 | ref_cnstrnt_notfound | Name of Referential Constraint not defined in constraints table.<br>Имя ссылочного ограничения не определено в таблице ограничений. |
| -660 | 335544533 | foreign_key_notfound | Non-existent PRIMARY or UNIQUE KEY specified for FOREIGN KEY.<br>Не существует первичного или уникального ключа, указанного для внешнего ключа. |
| -292 | 335544534 | ref_cnstrnt_update | Cannot update constraints (RDB$REF_CONSTRAINTS).<br>Нельзя изменять ограничения (RDB$REF_CONSTRAINTS). |
| -293 | 335544535 | check_cnstrnt_update | Cannot update constraints (RDB$CHECK_CONSTRAINTS).<br>Нельзя изменять ограничения (RDB$REF_CONSTRAINTS). |
| -294 | 335544536 | check_cnstrnt_del | Cannot delete CHECK constraint entry (RDB$CHECK_CONSTRAINTS)<br>Нельзя удалять запись ограничения CHECK (RDB$REF_CONSTRAINTS). |
| -618 | 335544537 | integ_index_seg_del | Cannot delete index segment used by an Integrity Constraint<br>Невозможно удалить сегмент индекса, используемого в ограничении целостности. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -618 | 335544538 | integ_index_seg_mod | Cannot update index segment used by an Integrity Constraint<br>Невозможно изменить сегмент индекса, используемого в ограничении целостности. |
| -616 | 335544539 | integ_index_del | Cannot delete index used by an Integrity Constraint<br>Невозможно удалить индекс, используемый в ограничении целостности. |
| -616 | 335544540 | integ_index_mod | Cannot modify index used by an Integrity Constraint<br>Невозможно изменить индекс, используемый в ограничении целостности. |
| -616 | 335544541 | check_trig_del | Cannot delete trigger used by a CHECK Constraint<br>Невозможно удалить триггер, используемый в ограничении CHECK. |
| -617 | 335544542 | check_trig_update | Cannot update trigger used by a CHECK Constraint<br>Невозможно изменить триггер, используемый в ограничении CHECK. |
| -616 | 335544543 | cnstrnt_fld_del | Cannot delete column being used in an Integrity Constraint.<br>Невозможно удалить столбец, используемый в ограничении целостности. |
| -617 | 335544544 | cnstrnt_fld_rename | Cannot rename column being used in an Integrity Constraint.<br>Невозможно переименовать столбец, используемый в ограничении целостности. |
| -295 | 335544545 | rel_cnstrnt_update | Cannot update constraints (RDB$RELATION_CONSTRAINTS).<br>Нельзя изменять ограничения (RDB$RELATION_CONSTRAINTS). |
| -150 | 335544546 | constaint_on_view | Cannot define constraints on views<br>Нельзя определить ограничения для представлений. |
| -296 | 335544547 | invld_cnstrnt_type | internal Firebird consistency check (invalid RDB$CONSTRAINT_TYPE)<br>Внутренняя ошибка программного обеспечения на согласованность (неверный тип RDB$CONSTRAINT_TYPE). |
| -831 | 335544548 | primary_key_exists | Attempt to define a second PRIMARY KEY for the same table<br>Попытка определения второго первичного ключа для той же таблицы. |
| -607 | 335544549 | systrig_update | cannot modify or erase a system trigger<br>Невозможно изменить или удалить системный триггер. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -552 | 335544550 | not_rel_owner | only the owner of a table may reassign ownership<br>Не используется. |
| -204 | 335544551 | grant_obj_notfound | could not find object for GRANT<br>Невозможно найти объект для GRANT. |
| -205 | 335544552 | grant_fld_notfound | could not find column for GRANT<br>Невозможно найти столбец для GRANT. |
| -552 | 335544553 | grant_nopriv | user does not have GRANT privileges for operation<br>Пользователь не имеет назначенных привилегий для операции. |
| -84 | 335544554 | nonsql_security_rel | object has non-SQL security class defined<br>Для объекта определен класс безопасности, не являющийся классом SQL. |
| -84 | 335544555 | nonsql_security_fld | column has non-SQL security class defined<br>Для столбца определен класс безопасности, не являющийся классом SQL. |
| -255 | 335544556 | wal_cache_err | Write-ahead Log without shared cache configuration not allowed<br>Не используется. |
| -902 | 335544557 | shutfail | database shutdown unsuccessful<br>Неуспешный останов базы данных. |
| -297 | 335544558 | check_constraint | Operation violates CHECK constraint @1 on view or table @2<br>Операция нарушает ограничение CHECK @1 для представления или таблицы @2. |
| -901 | 335544559 | bad_svc_handle | invalid service handle<br>Неверный дескриптор сервиса. |
| -838 | 335544560 | shutwarn | database @1 shutdown in @2 seconds<br>Не используется. |
| -901 | 335544561 | wrospbver | wrong version of service parameter block<br>Неверная версия блока параметра сервиса. |
| -901 | 335544562 | bad_spb_form | unrecognized service parameter block<br>Нераспознанный блок параметров сервиса. |
| -901 | 335544563 | svcnotdef | service @1 is not defined<br>Сервис @1 не определен. |
| -902 | 335544564 | no_jrn | long-term journaling not enabled<br>Не используется. |
| -314 | 335544565 | transliteration_failed | Cannot transliterate character between character sets<br>Невозможно выполнить транслитерацию символов между наборами символов. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -257 | 335544566 | start_cm_for_wal | WAL defined; Cache Manager must be started first<br>Не используется. |
| -258 | 335544567 | wal_ovflow_log_required | Overflow log specification required for round-robin log<br>Не используется. |
| -204 | 335544568 | text_subtype | Implementation of text subtype @1 not located.<br>Реализация текстового подтипа @1 не обнаружена. |
| -902 | 335544569 | dsql_error | Dynamic SQL Error<br>Ошибка динамического SQL. |
| -104 | 335544570 | dsql_command_err | Invalid command<br>Неверная команда. |
| -103 | 335544571 | dsql_constant_err | Data type for constant unknown<br>Неизвестный тип данных для константы. |
| -504 | 335544572 | dsql_cursor_err | Invalid cursor reference<br>Недопустимая ссылка на курсор. |
| -204 | 335544573 | dsql_datatype_err | Data type unknown<br>Неизвестный тип данных. |
| -502 | 335544574 | dsql_decl_err | Invalid cursor declaration<br>Неверная декларация курсора. |
| -510 | 335544575 | dsql_cursor_update_err | Cursor @1 is not updatable<br>Курсор @1 является не обновляемым. |
| -502 | 335544576 | dsql_cursor_open_err | Attempt to reopen an open cursor<br>Попытка переоткрыть открытый курсор. |
| -501 | 335544577 | dsql_cursor_close_err | Attempt to reclose a closed cursor<br>Попытка перезакрыть закрытый курсор. |
| -206 | 335544578 | dsql_field_err | Column unknown<br>Столбец неизвестен. |
| -104 | 335544579 | dsql_internal_err | Internal error<br>Внутренняя ошибка. |
| -204 | 335544580 | dsql_relation_err | Table unknown<br>Неизвестная таблица. |
| -204 | 335544581 | dsql_procedure_err | Procedure unknown<br>Неизвестная процедура. |
| -518 | 335544582 | dsql_request_err | Request unknown<br>Запрос неизвестен. |
| -804 | 335544583 | dsql_sqlda_err | SQLDA error<br>Ошибка SQLDA |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -804 | 335544584 | dsql_var_count_err | Count of read-write columns does not equal count of values<br>Количество столбцов для чтения/записи не равно количеству значений. |
| -826 | 335544585 | dsql_stmt_handle | Invalid statement handle<br>Не используется. |
| -804 | 335544586 | dsql_function_err | Function unknown<br>Неизвестная функция. |
| -206 | 335544587 | dsql_blob_err | Column is not a BLOB<br>Не используется. |
| -204 | 335544588 | collation_not_found | COLLATION @1 for CHARACTER SET @2 is not defined<br>Тип сортировки @1 для набора символов @2 не определен. |
| -204 | 335544589 | collation_not_for_-charset | COLLATION @1 is not valid for specified CHARACTER SET<br>Тип сортировки @1 неверный для указанного набора символов. |
| -104 | 335544590 | dsql_dup_option | Option specified more than once<br>Не используется. |
| -104 | 335544591 | dsql_tran_err | Unknown transaction option<br>Не используется. |
| -104 | 335544592 | dsql_invalid_array | Invalid array reference<br>Неверная ссылка на массив. |
| -604 | 335544593 | dsql_max_arr_dim_-exceeded | Array declared with too many dimensions<br>Объявлен массив со слишком большой размерностью. |
| -604 | 335544594 | dsql_arr_range_error | Illegal array dimension range<br>Неверный диапазон размерности массива. |
| -204 | 335544595 | dsql_trigger_err | Trigger unknown<br>Не используется. |
| -206 | 335544596 | dsql_subselect_err | Subselect illegal in this context<br>Не используется. |
| -531 | 335544597 | dsql_crdb_prepare_err | Cannot prepare a CREATE DATABASE/SCHEMA statement<br>Невозможно подготовить к выполнению оператор CREATE DATABASE/SCHEMA. |
| -157 | 335544598 | specify_field_err | must specify column name for view select expression<br>Требуется задать имя столбца для выражения SELECT в представлении. |
| -158 | 335544599 | num_field_err | number of columns does not match select list<br>Номера столбцов не соответствуют списку выборки SELECT. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -806 | 335544600 | col_name_err | Only simple column names permitted for VIEW WITH CHECK OPTION<br>Только простые имена столбцов допустимы в предложении VIEW WITH CHECK OPTION. |
| -807 | 335544601 | where_err | No WHERE clause for VIEW WITH CHECK OPTION<br>Нет предложения WHERE для опции VIEW WITH CHECK. |
| -808 | 335544602 | table_view_err | Only one table allowed for VIEW WITH CHECK OPTION<br>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<br>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<br>Количество столбцов внешнего ключа не соответствует первичному ключу. |
| -810 | 335544605 | subquery_err | No subqueries permitted for VIEW WITH CHECK OPTION<br>Нет подзапросов разрешенных для VIEW WITH CHECK OPTION. |
| -833 | 335544606 | expression_eval_err | expression evaluation not supported<br>Результат вычисления выражения не поддерживается. |
| -599 | 335544607 | node_err | gen.c: node not supported<br>Не используется. |
| -104 | 335544608 | command_end_err | Unexpected end of command<br>Неверное завершение команды. |
| -901 | 335544609 | index_name | INDEX @1<br>Индекс @1. |
| -901 | 335544610 | exception_name | EXCEPTION @1<br>Исключение @1. |
| -901 | 335544611 | field_name | COLUMN @1<br>Столбец @1. |
| -104 | 335544612 | token_err | Token unknown<br>Неизвестный синтаксический элемент. |
| -901 | 335544613 | union_err | union not supported<br>Не используется. |
| -901 | 335544614 | dsql_construct_err | Unsupported DSQL construct<br>Не поддерживаемая конструкция DSQL |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -830 | 335544615 | field_aggregate_err | column used with aggregate<br>Не используется. |
| -829 | 335544616 | field_ref_err | invalid column reference<br>Не используется. |
| -208 | 335544617 | order_by_err | invalid ORDER BY clause<br>Неверное предложение ORDER BY. |
| -171 | 335544618 | return_mode_err | Return mode by value not allowed for this data type<br>Вариант возвращаемого значения недоступен для этого типа данных. |
| -170 | 335544619 | extern_func_err | External functions cannot have more than 10 parameters<br>Внешняя функция не может иметь более 10 параметров. |
| -204 | 335544620 | alias_conflict_err | alias @1 conflicts with an alias in the same statement<br>Псевдоним @1 конфликтует с псевдонимом в том же операторе. |
| -204 | 335544621 | procedure_conflict_-error | alias @1 conflicts with a procedure in the same statement<br>Алиас @1 конфликтует в процедурой в том же выражении. |
| -204 | 335544622 | relation_conflict_err | alias @1 conflicts with a table in the same statement<br>Алиас @2 конфликтует с таблицей в том же выражении. |
| -901 | 335544623 | dsql_domain_err | Illegal use of keyword VALUE<br>Неверное использование ключевого слова VALUE. |
| -663 | 335544624 | idx_seg_err | segment count of 0 defined for index @1<br>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<br>Имя узла не разрешено в имени вторичного фала, файла оперативной копии, в кэше или в имени файла протокола. |
| -901 | 335544626 | table_name | TABLE @1<br>Таблица @1. |
| -901 | 335544627 | proc_name | PROCEDURE @1<br>Процедура @1 |
| -660 | 335544628 | idx_create_err | cannot create index @1<br>Невозможно создать индекс @1. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -259 | 335544629 | wal_shadow_err | Write-ahead Log with shadowing configuration not allowed<br>Не используется. |
| -616 | 335544630 | dependency | there are @1 dependencies<br>Есть @1 зависимостей. |
| -663 | 335544631 | idx_key_err | too many keys defined for index @1<br>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<br>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<br>Номер оперативной копии должен быть положительным целым числом. |
| -104 | 335544634 | dsql_token_unk_err | Token unknown - line @1, column @2<br>Неизвестный синтаксический элемент - строка @1, символ @2. |
| -204 | 335544635 | dsql_no_relation_alias | there is no alias or table named @1 at this scope level<br>Не существует указанного псевдонима или таблицы с именем @1 на этом уровне видимости. |
| -204 | 335544636 | indexname | there is no index @1 for table @2<br>Не существует индекса @1 для таблицы @2. |
| -281 | 335544637 | no_stream_plan | table @1 is not referenced in plan<br>Таблица @1 не упоминается в плане. |
| -282 | 335544638 | stream_twice | table @1 is referenced more than once in plan; use aliases to distinguish<br>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<br>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<br>Неверное использование набора символов или порядка сортировки. |
| -901 | 335544641 | dsql_domain_not_found | Specified domain or source column @1 does not exist<br>Указанный домен или исходный столбец @1 не существует. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -284 | 335544642 | index_unused | index @1 cannot be used in the specified plan<br>Индекс @1 не может быть использован в указанном плане. |
| -282 | 335544643 | dsql_self_join | the table @1 is referenced twice; use aliases to differentiate<br>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<br>Попытка сделать выборку, начиная до первой записи в потоке записи. |
| -595 | 335544645 | stream_crack | the current position is on a crack<br>Не используется. |
| -601 | 335544646 | db_or_file_exists | database or file exists<br>Не используется. |
| -401 | 335544647 | invalid_operator | invalid comparison operator for find operation<br>Не используется. |
| -924 | 335544648 | conn_lost | Connection lost to pipe server<br>Не используется. |
| -835 | 335544649 | bad_checksum | bad checksum<br>Не используется. |
| -689 | 335544650 | page_type_err | wrong page type<br>Неверный тип страницы. |
| -816 | 335544651 | ext_readonly_err | Cannot insert because the file is readonly or is on a read only medium.<br>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<br>Множество строк в одиночном операторе SELECT. |
| -902 | 335544653 | psw_attach | cannot attach to password database<br>Невозможно соединиться с базой данных пароля. |
| -902 | 335544654 | psw_start_trans | cannot start transaction for password database<br>Невозможно стартовать транзакцию для базы данных пароля. |
| -827 | 335544655 | invalid_direction | invalid direction for find operation<br>Не используется. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -901 | 335544656 | dsql_var_conflict | variable @1 conflicts with parameter in same procedure<br>Переменная @1 конфликтует с параметром в той же процедуре. |
| -607 | 335544657 | dsql_no_blob_array | Array/BLOB/DATE data types not allowed in arithmetic<br>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<br>Не используется. |
| -282 | 335544659 | duplicate_base_table | table @1 is referenced twice in view; use an alias to distinguish<br>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<br>Представление @1 использует более одного раза базовую таблицу; используйте псевдонимы для различения. |
| -904 | 335544661 | index_root_page_full | cannot add index, index root page is full.<br>Невозможно добавить индекс, корневая страница индекса заполнена. |
| -204 | 335544662 | dsql_blob_type_unknown | BLOB SUB_TYPE @1 is not defined<br>Подтип BLOB @1 не определен. |
| -693 | 335544663 | req_max_clones_exceeded | Too many concurrent executions of the same request<br>Слишком много одновременных выполнений одного и того же запроса. |
| -637 | 335544664 | dsql_duplicate_spec | duplicate specification of @1 - not supported<br>Дублирование спецификации для @1 не поддерживается. |
| -803 | 335544665 | unique_key_violation | violation of PRIMARY or UNIQUE KEY constraint "@1" on table "@2"<br>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<br>Не используется. |
| -909 | 335544667 | drdb_completed_with_errs | drop database completed with errors<br>Удаление базы данных завершилось с ошибками. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -84 | 335544668 | dsql_procedure_use_err | procedure @1 does not return any values<br>Процедура @1 не возвращает никакого значения. |
| -313 | 335544669 | dsql_count_mismatch | count of column list and variable list do not match<br>Количество столбцов и переменных в списке не соответствует. |
| -685 | 335544670 | blob_idx_err | attempt to index BLOB column in index @1<br>Попытка индексации BLOB столбца в индексе @1. |
| -685 | 335544671 | array_idx_err | attempt to index array column in index @1<br>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?)<br>Слишком много столбцов ключей обнаружено для индекса @1 (неверные имена столбцов?). |
| -901 | 335544673 | no_delete | cannot delete<br>Удаление невозможно. |
| -616 | 335544674 | del_last_field | last column in a table cannot be deleted<br>Последний столбец в таблице не может быть удален. |
| -901 | 335544675 | sort_err | sort error<br>Ошибка сортировки. |
| -904 | 335544676 | sort_mem_err | sort error: not enough memory<br>Ошибка сортировки: нет достаточного объема памяти. |
| -841 | 335544677 | version_err | too many versions<br>Слишком много версий. |
| -828 | 335544678 | inval_key_posn | invalid key position<br>Неверная позиция ключа. |
| -690 | 335544679 | no_segments_err | segments not allowed in expression index @1<br>Сегменты не допускаются в выражении индекса @1. |
| -600 | 335544680 | crrp_data_err | sort error: corruption in data structure<br>Не используется. |
| -691 | 335544681 | rec_size_err | new record size of @1 bytes is too big<br>Новый размер записи в @1 байт является слишком большим. |
| -605 | 335544682 | dsql_field_ref | Inappropriate self-reference of column<br>Недопустимая ссылка столбца на самого себя. |
| -904 | 335544683 | req_depth_exceeded | request depth exceeded. (Recursive definition?)<br>Превышена глубина запроса (рекурсивное определение?) |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -694 | 335544684 | no_field_access | cannot access column @1 in view @2<br>Не могу получить доступ к столбцу @1 представления @2. |
| -162 | 335544685 | no_dbkey | dbkey not available for multi-table views<br>Ключ базы данных недоступен для многотабличных представлений. |
| -839 | 335544686 | jrn_format_err | journal file wrong format<br>Не используется. |
| -840 | 335544687 | jrn_file_full | intermediate journal file full<br>Не используется. |
| -519 | 335544688 | dsql_open_cursor_-request | The prepare statement identifies a prepare statement with an open cursor<br>Не используется. |
| -999 | 335544689 | ib_error | Firebird error<br>Не используется. |
| -260 | 335544690 | cache_redef | Cache redefined<br>Не используется. |
| -239 | 335544691 | cache_too_small | Insufficient memory to allocate page buffer cache<br>Недостаточно памяти для выделения кэша под буфер страницы. |
| -260 | 335544692 | log_redef | Log redefined<br>Не используется. |
| -239 | 335544693 | log_too_small | Log size too small<br>Не используется. |
| -239 | 335544694 | partition_too_small | Log partition size too small<br>Не используется. |
| -261 | 335544695 | partition_not_supp | Partitions not supported in series of log file specification<br>Не используется. |
| -261 | 335544696 | log_length_spec | Total length of a partitioned log must be specified<br>Не используется. |
| -842 | 335544697 | precision_err | Precision must be from 1 to 18<br>Точность должна быть в пределах от 1 до 18. |
| -842 | 335544698 | scale_nogt | Scale must be between zero and precision<br>Масштаб должен быть между нулем и значением точности. |
| -842 | 335544699 | expec_short | Short integer expected<br>Ожидается короткое целое. |
| -842 | 335544700 | expec_long | Long integer expected<br>Не используется. |
| -842 | 335544701 | expec_ushort | Unsigned short integer expected<br>Ожидается беззнаковое короткое целое. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -105 | 335544702 | escape_invalid | Invalid ESCAPE sequence<br>Неверная ESCAPE последовательность. |
| -901 | 335544703 | svcnoexe | service @1 does not have an associated executable<br>Не используется. |
| -901 | 335544704 | net_lookup_err | Failed to locate host machine.<br>Не удалось найти хост машины |
| -901 | 335544705 | service_unknown | Undefined service @1/@2.<br>Не используется. |
| -901 | 335544706 | host_unknown | The specified name was not found in the hosts file or Domain Name Services.<br>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<br>Пользователь не имеет назначенных привилегий на таблицу/ представление для операции. |
| -203 | 335544708 | dyn_fld_ambiguous | Ambiguous column reference.<br>Неоднозначная ссылка на столбец. |
| -104 | 335544709 | dsql_agg_ref_err | Invalid aggregate reference<br>Неверная ссылка на агрегат. |
| -282 | 335544710 | complex_view | navigational stream @1 references a view with more than one base table<br>Поток навигации @1 ссылается на представление с более чем одной базовой таблицей. |
| -901 | 335544711 | unprepared_stmt | Attempt to execute an unprepared dynamic SQL statement.<br>Попытка выполнения неподготовленного оператора динамического SQL. |
| -842 | 335544712 | expec_positive | Positive value expected<br>Ожидается положительное значение. |
| -804 | 335544713 | dsql_sqlda_value_err | Incorrect values within SQLDA structure<br>Некорректные значения в SQLDA структуре. |
| -104 | 335544714 | invalid_array_id | invalid blob id<br>Неверный идентификатор BLOB. |
| -816 | 335544715 | extfile_uns_op | Operation not supported for EXTERNAL FILE table @1<br>Операция не поддерживается для внешней таблицы @1. |
| -901 | 335544716 | svc_in_use | Service is currently busy: @1<br>Сервис в настоящий момент занят: @1 |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -902 | 335544717 | err_stack_limit | stack size insufficent to execute current request<br>Не используется. |
| -827 | 335544718 | invalid_key | Invalid key for find operation<br>Неверный ключ для операции поиска. |
| -901 | 335544719 | net_init_error | Error initializing the network software.<br>Ошибка инициализации сетевого программного обеспечения. |
| -901 | 335544720 | loadlib_failure | Unable to load required library @1.<br>Не используется. |
| -902 | 335544721 | network_error | Unable to complete network request to host "@1".<br>Невозможно завершить сетевой запрос на хост "@1" |
| -902 | 335544722 | net_connect_err | Failed to establish a connection.<br>Ошибка при установлении соединения. |
| -902 | 335544723 | net_connect_listen_err | Error while listening for an incoming connection.<br>Ошибка при прослушивании входного соединения. |
| -902 | 335544724 | net_event_connect_err | Failed to establish a secondary connection for event processing.<br>Ошибка при установлении вторичного соединения для обработки события |
| -902 | 335544725 | net_event_listen_err | Error while listening for an incoming event connection request.<br>Ошибка при прослушивании запроса события соединения. |
| -902 | 335544726 | net_read_err | Error reading data from the connection.<br>Ошибка чтения данных из соединения. |
| -902 | 335544727 | net_write_err | Error writing data to the connection.<br>Ошибка записи данных в соединение |
| -616 | 335544728 | integ_index_deactivate | Cannot deactivate index used by an integrity constraint<br>Невозможно деактивировать индекс, используемый в ограничении целостности. |
| -616 | 335544729 | integ_deactivate_-primary | Cannot deactivate index used by a PRIMARY/UNIQUE constraint<br>Невозможно деактивировать индекс используемый в ограничении первичного/уникального ключа. |
| -104 | 335544730 | cse_not_supported | Client/Server Express not supported in this release<br>Не используется. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -901 | 335544731 | tra_must_sweep | -<br>Не используется. |
| -902 | 335544732 | unsupported_network_-drive | Access to databases on file servers is not supported.<br>Не используется. |
| -902 | 335544733 | io_create_err | Error while trying to create file<br>Ошибка ввода-вывода при попытке создания файла |
| -902 | 335544734 | io_open_err | Error while trying to open file<br>Ошибка при попытке открытия файла. |
| -902 | 335544735 | io_close_err | Error while trying to close file<br>Не используется. |
| -902 | 335544736 | io_read_err | Error while trying to read from file<br>Ошибка при попытке чтения из файла. |
| -902 | 335544737 | io_write_err | Error while trying to write to file<br>Ошибка при попытке записи в файл. |
| -902 | 335544738 | io_delete_err | Error while trying to delete file<br>Ошибка при попытке удаления файла. |
| -902 | 335544739 | io_access_err | Error while trying to access file<br>Ошибка при попытке доступа к файлу. |
| -901 | 335544740 | udf_exception | A fatal exception occurred during the execution of a user defined function.<br>Не используется. |
| -901 | 335544741 | lost_db_connection | connection lost to database<br>Потеряно соединение с базой данных. |
| -901 | 335544742 | no_write_user_priv | User cannot write to RDB$USER_PRIVILEGES<br>Пользователь не может писать в таблицу RDB$USER_PRIVILEGES |
| -104 | 335544743 | token_too_long | token size exceeds limit<br>Размер синтаксического элемента превышает допустимый предел. |
| -906 | 335544744 | max_att_exceeded | Maximum user count exceeded. Contact your database administrator.<br>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.<br>Ваше регистрационное имя @1 совпадает с именем роли SQL. Уточните у вашего администратора базы данных допустимое регистрационное имя Firebird. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -607 | 335544746 | reftable_requires_pk | "REFERENCES table" without "(column)" requires PRIMARY KEY on referenced table<br>"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.<br>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.<br>Не используется. |
| -85 | 335544749 | usrname_required | A username is required for this operation.<br>Для этой операции требуется имя пользователя. |
| -85 | 335544750 | password_required | A password is required for this operation<br>Для этой операции требуется пароль. |
| -85 | 335544751 | bad_protocol | The network protocol specified is invalid<br>Указан неверный сетевой протокол. |
| -85 | 335544752 | dup_usrname_found | A duplicate user name was found in the security database<br>Не используется. |
| -85 | 335544753 | usrname_not_found | The user name specified was not found in the security database<br>Указанное имя пользователя не найдено в базе данных безопасности. |
| -85 | 335544754 | error_adding_sec_record | An error occurred while attempting to add the user.<br>Обнаружена ошибка при попытке добавления пользователя. |
| -85 | 335544755 | error_modifying_sec_-record | An error occurred while attempting to modify the user record.<br>Обнаружена ошибка при попытке изменения записи пользователя. |
| -85 | 335544756 | error_deleting_sec_-record | An error occurred while attempting to delete the user record.<br>Обнаружена ошибка при попытке удаления записи пользователя. |
| -85 | 335544757 | error_updating_sec_db | An error occurred while updating the security database.<br>Обнаружена ошибка при изменении базы данных безопасности. |
| -904 | 335544758 | sort_rec_size_err | sort record size of @1 bytes is too big<br>Размер записи сортировки в @1 байт слишком велик |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -204 | 335544759 | bad_default_value | can not define a not null column with NULL as default value<br>Нельзя определять непустой столбец (NOT NULL) вместе со значением по умолчанию NULL. |
| -204 | 335544760 | invalid_clause | invalid clause -- '@1'<br>Неверное предложение - '@1'. |
| -904 | 335544761 | too_many_handles | too many open handles to database<br>Слишком много открытых дескрипторов базы данных. |
| -904 | 335544762 | optimizer_blk_exc | size of optimizer block exceeded<br>Превышен размер блока оптимизатора. |
| -104 | 335544763 | invalid_string_constant | a string constant is delimited by double quotes<br>Строковая константа заключена в двойные кавычки. |
| -104 | 335544764 | transitional_date | DATE must be changed to TIMESTAMP<br>DATE должно быть изменено в TIMESTAMP. |
| -817 | 335544765 | read_only_database | attempted update on read-only database<br>Попытка изменения базы данных только для чтения. |
| -817 | 335544766 | must_be_dialect_2_and_up | SQL dialect @1 is not supported in this database<br>Не используется. |
| -901 | 335544767 | blob_filter_exception | A fatal exception occurred during the execution of a blob filter.<br>Не используется. |
| -901 | 335544768 | exception_access_violation | Access violation. The code attempted to access a virtual address without privilege to do so.<br>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.<br>Не используется. |
| -901 | 335544770 | exception_array_bounds_exceeded | Array bounds exceeded. The code attempted to access an array element that is out of bounds.<br>Превышены границы размеров массива. Код пытается получить доступ к элементу массива, который находится за пределами его границ. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -901 | 335544771 | exception_float_denormal_operand | Float denormal operand. One of the floating-point operands is too small to represent a standard float value.<br>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.<br>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.<br>Не используется. |
| -901 | 335544774 | exception_float_invalid_operand | Floating-point invalid operand. An indeterminant error occurred during a floating-point operation.<br>Не используется. |
| -901 | 335544775 | exception_float_overflow | Floating-point overflow. The exponent of a floating-point operation is greater than the magnitude allowed.<br>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.<br>Не используется. |
| -901 | 335544777 | exception_float_underflow | Floating-point underflow. The exponent of a floating-point operation is less than the magnitude allowed.<br>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.<br>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.<br>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.<br>Не используется. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -901 | 335544781 | exception_stack_-overflow | Stack overflow. The resource requirements of the runtime stack have exceeded the memory available to it.<br>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.<br>Не используется. |
| -901 | 335544783 | exception_sigill | Illegal Instruction. The Code attempted to perfrom an illegal operation.<br>Неверная операция. Код пытается выполнить неверную операцию. |
| -901 | 335544784 | exception_sigbus | Bus Error. The Code caused a system bus error.<br>Не используется. |
| -901 | 335544785 | exception_sigfpe | Floating Point Error. The Code caused an Arithmetic Exception or a floating point exception.<br>Не используется. |
| -901 | 335544786 | ext_file_delete | Cannot delete rows from external files.<br>Невозможно удалять строки из внешних файлов. |
| -901 | 335544787 | ext_file_modify | Cannot update rows in external files.<br>Невозможно изменять строки во внешних файлах. |
| -901 | 335544788 | adm_task_denied | Unable to perform operation. You must be either SYSDBA or owner of the database<br>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<br>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.<br>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.<br>Не используется. |
| -904 | 335544792 | service_att_err | Cannot attach to services manager<br>Невозможно подключиться к менеджеру сервисов. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -817 | 335544793 | ddl_not_allowed_by_db_-sql_dial | Metadata update statement is not allowed by the current database SQL dialect @1<br>Оператор изменения метаданных недопустим в текущем диалекте SQL базы данных @1. |
| -901 | 335544794 | cancelled | operation was cancelled<br>Операция была отменена. |
| -902 | 335544795 | unexp_spb_form | unexpected item in service parameter block, expected @1<br>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<br>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<br>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.<br>Не используется. |
| -904 | 335544799 | svc_name_missing | The service name was not specified.<br>Не было указано имя сервиса. |
| -204 | 335544800 | too_many_contexts | Too many Contexts of Relation/Procedure/Views. Maximum allowed is 256<br>Слишком большой контекст в отношении/процедуре/представлении. Допустимо максимум 256. |
| -901 | 335544801 | datype_notsup | data type not supported for arithmetic<br>Тип данных не поддерживается для арифметических операций |
| 501 | 335544802 | dialect_reset_warning | Database dialect being changed from 3 to 1<br>Диалог базы данных изменился с 3 на 1 |
| -901 | 335544803 | dialect_not_changed | Database dialect not changed.<br>Диалект базы данных не был изменен. |
| -901 | 335544804 | database_create_failed | Unable to create database @1<br>Невозможно создать базу данных @1. |
| -901 | 335544805 | inv_dialect_specified | Database dialect @1 is not a valid dialect.<br>Диалект базы данных @1 не является допустимым диалектом. |
| -901 | 335544806 | valid_db_dialects | Valid database dialects are @1.<br>Допустимыми диалектами базы данных являются @1. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| 300 | 335544807 | sqlwarn | SQL warning code = @1<br>Код SQL предупреждения = @1 |
| 301 | 335544808 | dtype_renamed | DATE data type is now called TIMESTAMP<br>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.<br>Не используется. |
| -833 | 335544810 | date_range_exceeded | value exceeds the range for valid dates<br>Значение превышает диапазон допустимых дат. |
| -901 | 335544811 | inv_client_dialect_-specified | passed client dialect @1 is not a valid dialect.<br>Переданный клиентом диалект @1 не является верным диалектом |
| -901 | 335544812 | valid_client_dialects | Valid client dialects are @1.<br>Допустимыми клиентскими диалектами являются @1. |
| -904 | 335544813 | optimizer_between_err | Unsupported field type specified in BETWEEN predicate.<br>Не используется. |
| -901 | 335544814 | service_not_supported | Services functionality will be supported in a later version of the product<br>Функциональность сервисов будет поддерживаться на более поздних версиях продукта. |
| -607 | 335544815 | generator_name | GENERATOR @1<br>Генератор @1. |
| -607 | 335544816 | udf_name | UDF @1. |
| -204 | 335544817 | bad_limit_param | Invalid parameter to FETCH or FIRST. Only integers >= 0 are allowed.<br>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.<br>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.<br>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<br>Невозможно найти точку сохранения с именем @1 в контексте транзакции. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -104 | 335544821 | dsql_column_pos_err | Invalid column position used in the @1 clause<br>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<br>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<br>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)<br>Неверное выражение в @1 (не содержится ни в агрегатной функции, ни в предложении GROUP BY) |
| -104 | 335544825 | dsql_agg_having_err | Invalid expression in the @1 (neither an aggregate function nor a part of the GROUP BY clause)<br>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<br>Вложенные агрегатные или оконные функции недопустимы. |
| -904 | 335544827 | exec_sql_invalid_arg | Invalid argument in EXECUTE STATEMENT - cannot convert to string<br>Не используется. |
| -904 | 335544828 | exec_sql_invalid_req | Wrong request type in EXECUTE STATEMENT '@1'<br>Не используется. |
| -904 | 335544829 | exec_sql_invalid_var | Variable type (position @1) in EXECUTE STATEMENT '@2' INTO does not match returned column type<br>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<br>Слишком много уровней рекурсии в EXECUTE STATEMENT. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -902 | 335544831 | conf_access_denied | Use of @1 at location @2 is not allowed by server configuration<br>Использование @1 в @2 не разрешено конфигурацией сервера |
| -904 | 335544832 | wrong_backup_state | Cannot change difference file name while database is in backup mode<br>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<br>Не используется. |
| -902 | 335544834 | cursor_not_open | Cursor is not open<br>Курсор не открыт. |
| -901 | 335544835 | bad_shutdown_mode | Target shutdown mode is invalid for database "@1"<br>Невозможно остановить базу данных "@1". |
| -802 | 335544836 | concat_overflow | Concatenation overflow. Resulting string cannot exceed 32765 bytes in length<br>Переполнение при конкатенации. Результирующая строка не может превышать 32 К6. |
| -204 | 335544837 | bad_substring_offset | Invalid offset parameter @1 to SUBSTRING. Only positive integers are allowed.<br>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<br>Ссылки на целевое значение внешнего ключа не существует. |
| -530 | 335544839 | foreign_key_references_present | Foreign key references are present for the record<br>Ссылки внешнего ключа присутствуют для записи. |
| -901 | 335544840 | no_update | cannot update<br>Невозможно обновить. |
| -902 | 335544841 | cursor_already_open | Cursor is already open<br>Курсор уже открыт. |
| -901 | 335544842 | stack_trace | @1 |
| -901 | 335544843 | ctx_var_not_found | Context variable @1 is not found in namespace @2<br>Контекстная переменная @1 не найдена в пространстве имен @2. |
| -901 | 335544844 | ctx_namespace_invalid | Invalid namespace name @1 passed to @2<br>Неверное имя пространства имен @1 передается в @2. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -901 | 335544845 | ctx_too_big | Too many context variables<br>Слишком много контекстных переменных. |
| -901 | 335544846 | ctx_bad_argument | Invalid argument passed to @1<br>Неверный аргумент передан в @1 |
| -901 | 335544847 | identifier_too_long | BLR syntax error. Identifier @1... is too long<br>Ошибка синтаксиса BLR. Идентификатор @1 является слишком большим |
| -836 | 335544848 | except2 | exception @1<br>Не используется. |
| -104 | 335544849 | malformed_string | Malformed string<br>Искаженная (некорректная) строка. |
| -170 | 335544850 | prc_out_param_mismatch | Output parameter mismatch for procedure @1<br>Несоответствующие выходные параметры для процедуры @1. |
| -104 | 335544851 | command_end_err2 | Unexpected end of command - line @1, column @2<br>Неожиданное завершение команды - строка @1, символ @2. |
| -904 | 335544852 | partner_idx_incompat_-type | partner index segment no @1 has incompatible data type<br>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.<br>Неверный параметр длины @1 для SUBSTRING. Отрицательные целые числа не доступны. |
| -204 | 335544854 | charset_not_installed | CHARACTER SET @1 is not installed<br>Набор символов @1 не установлен. |
| -204 | 335544855 | collation_not_installed | COLLATION @1 for CHARACTER SET @2 is not installed<br>Сортировка @1 для набора символов @2 не установлена. |
| -902 | 335544856 | att_shutdown | connection shutdown<br>Соединение остановлено. |
| -904 | 335544857 | blobtoobig | Maximum BLOB size exceeded<br>Достигнут максимальный размер BLOB. |
| -607 | 335544858 | must_have_phys_field | Can't have relation with only computed fields or constraints<br>Не допустима таблица состоящая только из одних вычисляемых полей или ограничений. |
| -901 | 335544859 | invalid_time_precision | Time precision exceeds allowed range (0-@1)<br>Уровень точности времени (тип данных TIME) превышает допустимый диапазон (0 - @1) |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -413 | 335544860 | blob_convert_error | Unsupported conversion to target type BLOB (subtype @1)<br>Не используется. |
| -413 | 335544861 | array_convert_error | Unsupported conversion to target type ARRAY<br>Не поддерживается преобразование в массив. |
| -904 | 335544862 | record_lock_not_supp | Stream does not support record locking<br>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.<br>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<br>Превышено число допустимых транзакций. Выполнете бэкап и рестор, чтобы сделать вновь базу данных работоспособной. |
| -904 | 335544865 | field_disappeared | Column has been unexpectedly deleted<br>Столбец был неожиданно удален. |
| -901 | 335544866 | met_wrong_gtt_scope | @1 cannot depend on @2<br>@1 не может зависеть от @2. |
| -204 | 335544867 | subtype_for_internal_use | Blob sub_types bigger than 1 (text) are for internal use only<br>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)<br>Процедура @1 не является процедурой выборки (она не содержит оператор SUSPEND). |
| -901 | 335544869 | invalid_sort_datatype | Datatype @1 is not supported for sorting operation<br>Тип данных @1 не поддерживает сортировку. |
| -901 | 335544870 | collation_name | COLLATION @1<br>Порядок сортировки @1. |
| -901 | 335544871 | domain_name | DOMAIN @1<br>Домен @1. |
| -219 | 335544872 | domnotdef | domain @1 is not defined<br>Домен @1 не определен. |
| -171 | 335544873 | array_max_dimensions | Array data type can use up to @1 dimensions<br>Тип данных массив не может использовать свыше @1 размерностей. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -901 | 335544874 | max_db_per_trans_-allowed | A multi database transaction cannot span more than @1 databases<br>Не используется. |
| 0 | 335544875 | bad_debug_format | Bad debug info format<br>Неверный формат отладочной информации. |
| -901 | 335544876 | bad_proc_BLR | Error while parsing procedure @1's BLR<br>Ошибка во время разбора BLR процедуры @1. |
| -901 | 335544877 | key_too_big | index key too big<br>Не используется. |
| -904 | 335544878 | concurrent_transaction | concurrent transaction number is @1<br>Число конкурирующих транзакций @1 |
| -625 | 335544879 | not_valid_for_var | validation error for variable @1, value "@2"<br>Ошибка проверки данных для переменной @1, значение "@2". |
| -625 | 335544880 | not_valid_for | validation error for @1, value "@2"<br>Ошибка проверки данных для @1, значение "@2". |
| -820 | 335544881 | need_difference | Difference file name should be set explicitly for database on raw device<br>Файл дельты должен быть явно задан для базы данных на raw-устройстве. |
| -902 | 335544882 | long_login | Login name too long (@1 characters, maximum allowed @2)<br>Слишком длинное имя пользователя (@1 символов, максимально возможно @2) |
| -205 | 335544883 | fldnotdef2 | column @1 is not defined in procedure @2<br>Столбец @1 не определен в процедуре @2. |
| -105 | 335544884 | invalid_similar_pattern | Invalid SIMILAR TO pattern<br>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. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -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. |
| -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<br>Значение превышает допустимый интервал TIME |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -833 | 335544913 | datetime_range_exceeded | value exceeds the range for valid timestamps<br>Значение превышает допустимый интервал TIMESTAMP. |
| -802 | 335544914 | string_truncation | string right truncation<br>Строка усечена справа. |
| -802 | 335544915 | blob_truncation | blob truncation when converting to a string: length limit exceeded<br>При конвертации BLOB в строку произошло усечение: превышена допустимая длина. |
| -802 | 335544916 | numeric_out_of_range | numeric value is out of range<br>Число вышло за пределы диапазона. |
| -901 | 335544917 | shutdown_timeout | Firebird shutdown is still in progress after the specified timeout<br>Остановка Firebird продолжается по истечении указанного тайм-аута |
| -901 | 335544918 | att_handle_busy | Attachment handle is busy<br>Не используется. |
| -901 | 335544919 | bad_udf_freeit | Bad written UDF detected: pointer returned in FREE_IT function was not allocated by ib_util_malloc<br>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<br>Внешний провайдер '@1' не найден. |
| -901 | 335544921 | eds_connection | Execute statement error at @1 :2 Data source : @3<br>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<br>Ожидается имя параметра. |
| -901 | 335544925 | eds_unclosed_comment | Unclosed comment found near '@1'<br>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<br>Несоответствующие входные параметры. |
| -901 | 335544928 | eds_output_prm_mismatch | Output parameters mismatch<br>Несоответствующие выходные параметры. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -901 | 335544929 | eds_input_prm_not_set | Input parameter '@1' have no value set<br>Не установлен входной параметр '@1'. |
| -104 | 335544930 | too_big_blr | BLR stream length @1 exceeds implementation limit @2<br>Длина потока BLR @1 превышает возможный предел @2. |
| 0 | 335544931 | montabexh | Monitoring table space exhausted<br>Исчерпано пространство таблиц мониторинга. |
| -172 | 335544932 | modnotfound | module name or entrypoint could not be found<br>Имя модуля или точка входа не найдены. |
| -901 | 335544933 | nothing_to_cancel | nothing to cancel<br>Нечего отменять. |
| -901 | 335544934 | ibutil_not_loaded | ib_util library has not been loaded to deallocate memory returned by FREE_IT function<br>Не используется. |
| -904 | 335544935 | circular_computed | Cannot have circular dependencies with computed fields<br>В вычисляемых полях не может быть циклических зависимостей. |
| -902 | 335544936 | psw_db_error | Security database error<br>Ошибка в базе данных безопасности. |
| -833 | 335544937 | invalid_type_datetime_op | Invalid data type in DATE/TIME/TIMESTAMP addition or subtraction in add_datettime()<br>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<br>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<br>Только значение типа DATE может быть добавлено к TIME. |
| -833 | 335544940 | onlycansub_tstampfromtstamp | TIMESTAMP values can be subtracted only from another TIMESTAMP value<br>Значение типа TIMESTAMP можно вычесть только из другого TIMESTAMP значения. |
| -833 | 335544941 | onlyoneop_mustbe_tstamp | Only one operand can be of type TIMESTAMP<br>Только один операнд может быть типа TIMESTAMP. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -833 | 335544942 | invalid_extractpart_time | Only HOUR, MINUTE, SECOND and MILLISECOND can be extracted from TIME values<br>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<br>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<br>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<br>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<br>Первый аргумент для @1 должен быть целого типа или типа числа с плавающей точкой. |
| -833 | 335544947 | sysf_argviolates_uuidtype | Human readable UUID argument for @1 must be of string type<br>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<br>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"<br>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<br>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<br>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<br>Недопустимый тип данных в сложении с DATE/TIME/TIMESTAMP в @1. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -833 | 335544953 | sysf_invalid_addpart_dtime | Invalid part @1 to be added to a DATE/TIME/TIMESTAMP value in @2<br>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<br>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<br>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<br>Для совместного использования типа данных TIME с любым другим типом в @1 не разрешается использовать YEAR, MONTH, DAY и WEEK. |
| -833 | 335544957 | sysf_invalid_tstamptimediff | The result of TIME-TIMESTAMP or TIMESTAMP-TIME in @1 cannot be expressed in HOUR, MINUTE, SECOND or MILLISECOND.<br>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<br>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<br>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<br>Аргумент для @1 должен быть положительным. |
| -833 | 335544961 | sysf_basemustbe_positive | Base for @1 must be positive<br>Основание для @1 должно быть положительным. |
| -833 | 335544962 | sysf_argnmustbe_nonneg | Argument #@1 for @2 must be zero or positive<br>Аргумент #@1 для @2 должен быть больше или равен нулю. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -833 | 335544963 | sysf_argnmustbe_-positive | Argument #@1 for @2 must be positive<br>Аргумент #@1 для @2 должен быть больше нуля. |
| -833 | 335544964 | sysf_invalid_zeropowneg | Base for @1 cannot be zero if exponent is negative<br>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<br>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<br>Размер числа должен быть между -128 и 127 в @1. |
| -833 | 335544967 | sysf_argmustbe_nonneg | Argument for @1 must be zero or positive<br>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<br>16-ти байтный UUID для @1 должен быть строкового типа. |
| -833 | 335544969 | sysf_binuuid_wrongsize | Binary UUID argument for @2 must use @1 bytes<br>16-ти байтный UUID для @1 должен занимать @1 байт. |
| -902 | 335544970 | missing_required_spb | Missing required item @1 in service parameter block<br>Отсутствует необходимый элемент @1 в SPB. |
| -902 | 335544971 | net_server_shutdown | @1 server is shutdown<br>Сервер @1 остановлен. |
| -924 | 335544972 | bad_conn_str | Invalid connection string<br>Неверная строка подключения. |
| -901 | 335544973 | bad_epb_form | Unrecognized events block<br>Блок нераспознанных событий. |
| -902 | 335544974 | no_threads | Could not start first worker thread - shutdown server<br>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<br>Произошел таймаут во время ожидания вторичного соединения для обработки событий |
| -833 | 335544976 | sysf_argmustbe_nonzero | Argument for @1 must be different than zero<br>Аргумент для @1 должен отличаться от нуля. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -833 | 335544977 | sysf_argmustbe_range_inc1_1 | Argument for @1 must be in the range [-1, 1]<br>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<br>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]<br>Аргумент для @1 должен быть в диапазоне [-1;1]. |
| -104 | 335544980 | internal_rejected_params | Incorrect parameters provided to internal function @1<br>Неправильные параметры, предоставляемые внутренней функции @1. |
| -833 | 335544981 | sysf_fp_overflow | Floating point overflow in built-in function @1<br>Переполнение числа с плавающей точкой во встроенной функции @1. |
| -901 | 335544982 | udf_fp_overflow | Floating point overflow in result from UDF @1<br>Переполнение числа с плавающей точкой, возвращаемое UDF @1. |
| -901 | 335544983 | udf_fp_nan | Invalid floating point value returned by UDF @1<br>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<br>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<br>Недостаточно свободного места во временных каталогах. |
| -901 | 335544986 | eds_expl_tran_ctrl | Explicit transaction control is not allowed<br>Явное управление транзакциями запрещено. |
| -902 | 335544987 | no_trusted_spb | Use of TRUSTED switches in spb_command_line is prohibited<br>Использование переключателя TRUSTED в spb_command_line запрещено. |
| -901 | 335544988 | package_name | PACKAGE @1<br>Имя пакета @1. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -901 | 335544989 | cannot_make_not_null | Cannot make field @1 of table @2 NOT NULL because there are NULLs present<br>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<br>Это свойство больше не поддерживается. |
| -901 | 335544991 | view_name | VIEW @1<br>Имя представления @1. |
| -904 | 335544992 | lock_dir_access | Can not access lock files directory @1<br>Не удается получить доступ к каталогу файлов блокировок @1. |
| -901 | 335544993 | invalid_fetch_option | Fetch option @1 is invalid for a non-scrollable cursor<br>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<br>Невозможно выполнить функцию @1 нереализованного пакета @2. |
| -901 | 335544996 | proc_pack_not_-implemented | Cannot execute procedure @1 of the unimplemented package @2<br>Невозможно выполнить процедуру @1 нереализованного пакета @2. |
| -901 | 335544997 | eem_func_not_returned | External function @1 not returned by the external engine plugin @2<br>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<br>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<br>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<br>Несовместимая версия плагина @1 для внешнего движка @2 |
| -901 | 335545001 | eem_engine_notfound | External engine @1 not found<br>Движок для работы с внешними модулями @1 не найден. |
| -532 | 335545002 | attachment_in_use | Attachment is in use<br>Соединение используется. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -532 | 335545003 | transaction_in_use | Transaction is in use<br>Транзакция используется. |
| -901 | 335545004 | pman_cannot_load_plugin | Error loading plugin @1<br>Невозможно загрузить плагин @1. |
| -901 | 335545005 | pman_module_notfound | Loadable module @1 not found<br>Загружаемый модуль @ 1 не найден. |
| -901 | 335545006 | pman_entrypoint_-notfound | Standard plugin entrypoint does not exist in module @1<br>Стандартная точка входа плагина не существует в модуле @1. |
| -901 | 335545007 | pman_module_bad | Module @1 exists but can not be loaded<br>Модуль @1 существует, но не может быть загружен. |
| -901 | 335545008 | pman_plugin_notfound | Module @1 does not contain plugin @2 type @3<br>Модуль @1 не содержит плагин @2 типа @3. |
| -833 | 335545009 | sysf_invalid_trig_-namespace | Invalid usage of context namespace DDL_TRIGGER<br>Недопустимое использование пространства имен DDL_TRIGGER. |
| -901 | 335545010 | unexpected_null | Value is NULL but isNull parameter was not informed<br>Не используется. |
| -901 | 335545011 | type_notcompat_blob | Type @1 is incompatible with BLOB<br>Не используется. |
| -901 | 335545012 | invalid_date_val | Invalid date<br>Не используется. |
| -901 | 335545013 | invalid_time_val | Invalid time.<br>Не используется. |
| -901 | 335545014 | invalid_timestamp_val | Invalid timestamp.<br>Не используется. |
| -901 | 335545015 | invalid_index_val | Invalid index @1 in function @2<br>Неверный индекс @1 в функции @2. |
| -836 | 335545016 | formatted_exception | @1 |
| -532 | 335545017 | async_active | Asynchronous call is already running for this attachment<br>Асинхронный вызов уже запущен для этого подключения. |
| -901 | 335545018 | private_function | Function @1 is private to package @2<br>Функция @1 частная (private) в пакете @2. |
| -901 | 335545019 | private_procedure | Procedure @1 is private to package @2<br>Процедура @1 частная (private) в пакете @2. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -904 | 335545020 | request_outdated | Request can't access new records in relation @1 and should be recompiled<br>Не используется. |
| -901 | 335545021 | bad_events_handle | invalid events id (handle). |
| -104 | 335545022 | cannot_copy_stmt | Cannot copy statement @1<br>Невозможно скопировать оператор @1. |
| -104 | 335545023 | invalid_boolean_usage | Invalid usage of boolean expression<br>Недопустимое использование булевского выражения. |
| -833 | 335545024 | sysf_argscant_both_be_zero | Arguments for @1 cannot both be zero<br>Аргументы для @1 не могут быть равны нулю. |
| -901 | 335545025 | spb_no_id | missing service ID in spb<br>Отсутствует ID сервиса в SPB. |
| -901 | 335545026 | ee_blr_mismatch_null | External BLR message mismatch: invalid null descriptor at field @1<br>Не используется. |
| -901 | 335545027 | ee_blr_mismatch_length | External BLR message mismatch: length = @1, expected @2<br>Не используется. |
| -406 | 335545028 | ss_out_of_bounds | Subscript @1 out of bounds [@2, @3]<br>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 |
| -902 | 335545030 | protect_sys_tab | @1 operation is not allowed for system table @2<br>Операция @1 для системной таблицы @2 не разрешена. |
| -901 | 335545031 | libtommath_generic | Libtommath error code @1 in function @2<br>Код ошибки Libtommath @1 в функции @2. |
| -902 | 335545032 | wroblrver2 | unsupported BLR version (expected between @1 and @2, encountered @3)<br>Неподдерживаемая BLR версия (ожидается между @1 и @2, встречена @3). |
| -551 | 335545033 | trunc_limits | expected length @1, actual @2<br>Ожидаемая длина @1, фактическая @2 |
| -551 | 335545034 | info_access | Wrong info requested in isc_svc_query() for anonymous service<br>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<br>В запросе пользователя нет isc_info_svc_stdin, но поток службы запрашивает данные stdin. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -551 | 335545036 | svc_start_failed | Start request for anonymous service is impossible<br>Запуск анонимного сервиса невозможен. |
| -104 | 335545037 | svc_no_switches | All services except for getting server log require switches<br>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<br>Размер данных stdin больше, чем запрашивалось у клиента. |
| -104 | 335545039 | no_crypt_plugin | Crypt plugin @1 failed to load<br>Плагин шифрования @1 не удалось загрузить |
| -104 | 335545040 | cp_name_too_long | Length of crypt plugin name should not exceed @1 bytes<br>Длина имени плагина шифрования не должна превышать @1 байт |
| -901 | 335545041 | cp_process_active | Crypt failed - already crypting database<br>В шифровании отказано - база данных уже шифруется. |
| -901 | 335545042 | cp_already_crypted | Crypt failed - database is already in requested state<br>В шифровании отказано - база данных уже в требуемом состоянии. |
| -902 | 335545043 | decrypt_error | Missing crypt plugin, but page appears encrypted<br>Отсутствует плагин шифрования, но страница обозначена как зашифрованная. |
| -902 | 335545044 | no_providers | No providers loaded<br>Нет загруженных провайдеров |
| -104 | 335545045 | null_spb | NULL data with non-zero SPB length<br>NULL данные с ненулевой длиной SPB. |
| -833 | 335545046 | max_args_exceeded | Maximum (@1) number of arguments exceeded for function @2<br>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<br>Не используется. |
| -901 | 335545048 | ee_blr_mismatch_name_not_found | External BLR message mismatch: name @1 not found<br>Не используется. |
| -901 | 335545049 | bad_result_set | Invalid resultset interface<br>Недопустимый интерфейс набора данных. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -804 | 335545050 | wrong_message_length | Message length passed from user application does not match set of columns<br>Длина сообщения, переданная из пользовательского приложения, не соответствует набору столбцов. |
| -804 | 335545051 | no_output_format | Resultset is missing output format information<br>В наборе данных отсутствует информация о выходных данных. |
| -804 | 335545052 | item_finish | Message metadata not ready - item @1 is not finished<br>Метаданные сообщения не готовы - элемент @1 не завершен. |
| -902 | 335545053 | miss_config | Missing configuration file: @1<br>Отсутствует файл конфигурации: @1. |
| -902 | 335545054 | conf_line | @1: illegal line <@2><br>@1: недопустимая строка <@2>. |
| -902 | 335545055 | conf_include | Invalid include operator in @1 for <@2><br>Недопустимый оператор include в @1 для <@2>. |
| -902 | 335545056 | include_depth | Include depth too big<br>Глубина include слишком большая. |
| -902 | 335545057 | include_miss | File to include not found<br>Не найден файл для включения. |
| -552 | 335545058 | protect_ownership | Only the owner can change the ownership. |
| -901 | 335545059 | badvarnum | undefined variable number<br>Неопределенное число переменных |
| -902 | 335545060 | sec_context | Missing security context for @1<br>Отсутствует контекст безопасности для @1. |
| -902 | 335545061 | multi_segment | Missing segment @1 in multisegment connect block parameter<br>Отсутствует сегмент @1 в параметре блока многосегментного соединения. |
| -902 | 335545062 | login_changed | Different logins in connect and attach packets - client library error<br>Различные логины при подключении и пакетах соединения - ошибка клиентской библиотеки. |
| -902 | 335545063 | auth_handshake_limit | Exceeded exchange limit during authentication handshake<br>Превышен лимит обмена во время аутентификации |
| -902 | 335545064 | wirecrypt_incompatible | Incompatible wire encryption levels requested on client and server<br>Несовместимые уровни шифрования сессии, запрашиваемые на клиенте и сервере |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -902 | 335545065 | miss_wirecrypt | Client attempted to attach unencrypted but wire encryption is required<br>Клиент пытался подключиться незашифрованно, хотя требуется шифрование сессии. |
| -902 | 335545066 | wirecrypt_key | Client attempted to start wire encryption using unknown key @1<br>Клиент попытался запустить шифрование сессии с помощью неизвестного ключа @1. |
| -902 | 335545067 | wirecrypt_plugin | Client attempted to start wire encryption using unsupported plugin @1<br>Клиент попытался запустить шифрование сессии с помощью неподдерживаемого плагина @1. |
| -902 | 335545068 | secdb_name | Error getting security database name from configuration file<br>Ошибка при получении имени базы данных безопасности из файла конфигурации. |
| -902 | 335545069 | auth_data | Client authentication plugin is missing required data from server<br>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<br>Плагин аутентификации на клиенте ожидает @2 байтов @3 с сервера, хотя получил @1. |
| -901 | 335545071 | info_unprepared_stmt | Attempt to get information about an unprepared dynamic SQL statement.<br>Попытка получить информацию о неподготовленном DSQL операторе. |
| -901 | 335545072 | idx_key_value | Problematic key value is @1<br>Сомнительное значение ключа - @1. |
| -901 | 335545073 | forupdate_virtualtbl | Cannot select virtual table @1 for update WITH LOCK<br>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<br>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<br>Нельзя использовать FOR UPDATE WITH LOCK для выборки из временной таблицы @1. |
| -901 | 335545076 | cant_modify_sysobj | System @1 @2 cannot be modified<br>Системный объект @1 @2 не может быть изменен. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -901 | 335545077 | server_misconfigured | Server misconfigured - contact administrator please<br>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$ADMIN role<br>Оператор ALTER ROLE ... SET/DROP AUTO ADMIN MAPPING существует только для одной роли - RDB$ADMIN |
| -901 | 335545079 | map_already_exists | Mapping @1 already exists<br>Отображение @1 уже существует. |
| -901 | 335545080 | map_not_exists | Mapping @1 does not exist<br>Отображение @1 не существует. |
| -901 | 335545081 | map_load | @1 failed when loading mapping cache<br>@1 не удалось при загрузке кэша отображения. |
| -901 | 335545082 | map_aster | Invalid name <*> in authentication block<br>Недопустимое имя <*> в блоке аутентификации. |
| -901 | 335545083 | map_multi | Multiple maps found for @1<br>Найдено несколько отображений для @1. |
| -901 | 335545084 | map_undefined | Undefined mapping result - more than one different results found<br>Неопределенный результат отображения - найдено несколько разных результатов. |
| -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]<br>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<br>Попытка временно установить количество буферов меньше, чем @1. |
| -901 | 335545088 | map_nodb | Global mapping is not available when database @1 is not present<br>Глобальное отображение невозможно, если отсутствует база данных @1. |
| -901 | 335545089 | map_notable | Global mapping is not available when table RDB$MAP is not present in database @1<br>Глобальное отображение невозможно, если в базе данных @1 отсутствует таблица RDB$MAP. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -901 | 335545090 | miss_trusted_role | Your attachment has no trusted role<br>Ваше подключение не имеет доверительной роли (роли, полученной в результате доверительной аутентификации). |
| -901 | 335545091 | set_invalid_role | Role @1 is invalid or unavailable<br>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<br>Дублированный атрибут пользователя @1. |
| -901 | 335545094 | dyn_no_priv | There is no privilege for this operation<br>Для этой операции нет привилегий. |
| -901 | 335545095 | dsql_cant_grant_option | Using GRANT OPTION on @1 not allowed<br>Не разрешено использовать GRANT для @1. |
| -904 | 335545096 | read_conflict | read conflicts with concurrent update<br>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<br>Проверка на наличие привилегий CREATE DATABASE невозможна, если отсутствует база данных @1. |
| -901 | 335545099 | crdb_notable | CREATE DATABASE grants check is not possible when table RDB$DB_CREATORS is not present in database @1<br>Проверка на наличие привилегий CREATE DATABASE невозможна, если отсутствует таблица RDB$DB_CREATORS в базе данных @1. |
| -804 | 335545100 | interface_version_too_old | Interface @3 version too old: expected @1, found @2<br>Интерфейс @3 слишком устаревший: ожидается @1, найдено @2. |
| -170 | 335545101 | fun_param_mismatch | Input parameter mismatch for function @1<br>Несоответствие входных параметров для функции @1. |
| -901 | 335545102 | savepoint_backout_err | Error during savepoint backout - transaction invalidated<br>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<br>Домен, используемый в ограничении PRIMARY KEY таблицы @1, должен быть NOT NULL |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -204 | 335545104 | invalid_attachment_-charset | CHARACTER SET @1 cannot be used as a attachment character set<br>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<br>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<br>Ошибка при входе в систему. Пожалуйста, проверьте firebird.log. |
| -902 | 335545107 | already_opened | Database already opened with engine instance, incompatible with current<br>База данных уже открыта экземпляром сервера, несовместимым с текущим. |
| -902 | 335545108 | bad_crypt_key | Invalid crypt key @1<br>Неверный ключ шифрования @1. |
| -901 | 335545109 | encrypt_error | Page requires encyption but crypt plugin is missing<br>Страница требует шифрования, но плагин шифрования отсутствует. |
| -901 | 335546320 | bad_ext_file | Invalid external file format<br>Недопустимый формат внешнего файла. |
| -901 | 335546321 | wrong_adp_fields_def | Wrong set of adapter fields<br>Неверный набор полей адаптера |
| -901 | 335546322 | wrong_adp_field_name | Adapter field named "@1"is not supported<br>Поле адаптера с именем "@1" не поддерживается. |
| -901 | 335546323 | wrong_adp_field_type | Wrong type of adapter field "@1"<br>Неверный тип поля адаптера "@1". |
| -901 | 335546324 | bad_ext_record | Unknown record type in external file "@1" at offset @2<br>Неизвестный тип записи во внешнем файле "@1" при смещении @2. |
| -901 | 335546325 | bad_adp_type | Unknown adapter type "@1"<br>Неизвестный тип адаптера "@1". |
| -902 | 335546326 | dir_resolve_error | Cannot resolve directory name "@1"<br>Не удается определить имя каталога "@1". |
| -902 | 335546327 | read_file_error | Error reading file "@1": @2<br>Ошибка при чтении файла "@1": @2. |
| -902 | 335546328 | create_file_error | Error creating file "@1": @2<br>Ошибка при создании файла "@1": @2. |
| -902 | 335546329 | delete_file_error | Error deleting file "@1": @2<br>Ошибка при удалении файла "@1": @2. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -902 | 335546330 | blob_io_error | I/O error during "@1" operation for BLOB file "@2"<br>Ошибка ввода-вывода во время операции "@1" для BLOB файла "@2". |
| -902 | 335546331 | cant_connect_to_ldap | Cannot connect to LDAP server: @1<br>Не удается подключиться к LDAP серверу: @1. |
| -902 | 335546332 | cant_bind_to_ldap | Cannot bind to LDAP server with specified login and password<br>Не удается присоединиться к LDAP серверу с указанным логином и паролем. |
| -902 | 335546333 | no_ldap_init | libldap or liblber libraries were not loaded correctly<br>Библиотеки libldap или liblber не были загружены корректно. |
| -833 | 335546334 | sysf_ldap_attr | Cannot get user attribute from LDAP<br>Не удается получить атрибут пользователя из LDAP. |
| -901 | 335546335 | no_ext_file | Missing external file for adapter<br>Отсутствует внешний файл для адаптера. |
| -902 | 335546336 | no_gss_init | GSS library was not loaded correctly<br>GSS библиотека была загружена некорректно. |
| -901 | 335546337 | hash_error | Hash calculation error<br>Ошибка в вычислении хэша. |
| -901 | 335546338 | policy_already_exists | Policy @1 already exists<br>Не используется. |
| -901 | 335546339 | policy_not_exists | Policy @1 does not exist<br>Не используется. |
| -901 | 335546340 | user_not_exists | User @1 does not exist<br>Не используется. |
| -901 | 335546341 | policy_is_granted | Policy @1 is granted to a user<br>Не используется. |
| -901 | 335546342 | bad_trig_BLR | Error while parsing trigger @1's BLR<br>Ошибка при парсинге BLR триггера @1. |
| -901 | 335740929 | gfix_db_name | data base file name (@1) already given<br>Не используется. |
| -901 | 335740930 | gfix_invalid_sw | invalid switch @1<br>Не используется. |
| -901 | 335740932 | gfix_incmp_sw | incompatible switch combination<br>Не используется. |
| -901 | 335740933 | gfix_replay_req | replay log pathname required<br>Не используется. |
| -901 | 335740934 | gfix_pgbuf_req | number of page buffers for cache required<br>Не используется. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -901 | 335740935 | gfix_val_req | numeric value required<br>Не используется. |
| -901 | 335740936 | gfix_pval_req | positive numeric value required<br>Не используется. |
| -901 | 335740937 | gfix_trn_req | number of transactions per sweep required<br>Не используется. |
| -901 | 335740940 | gfix_full_req | "full" or "reserve" required<br>Не используется. |
| -901 | 335740941 | gfix_usrname_req | user name required<br>Не используется. |
| -901 | 335740942 | gfix_pass_req | password required<br>Не используется. |
| -901 | 335740943 | gfix_subs_name | subsystem name<br>Не используется. |
| -901 | 335740944 | gfix_wal_req | "wal" required<br>Не используется. |
| -901 | 335740945 | gfix_sec_req | number of seconds required<br>Не используется. |
| -901 | 335740946 | gfix_nval_req | numeric value between 0 and 32767 inclusive required<br>Не используется. |
| -901 | 335740947 | gfix_type_shut | must specify type of shutdown<br>Не используется. |
| -901 | 335740948 | gfix_retry | please retry, specifying an option<br>Не используется. |
| -901 | 335740951 | gfix_retry_db | please retry, giving a database name<br>Не используется. |
| -901 | 335740991 | gfix_exceed_max | internal block exceeds maximum size<br>Не используется. |
| -901 | 335740992 | gfix_corrupt_pool | corrupt pool<br>Не используется. |
| -901 | 335740993 | gfix_mem_exhausted | virtual memory exhausted<br>Не используется. |
| -901 | 335740994 | gfix_bad_pool | bad pool id<br>Не используется. |
| -901 | 335740995 | gfix_trn_not_valid | Transaction state @1 not in valid range.<br>Не используется. |
| -901 | 335741012 | gfix_unexp_eoi | unexpected end of input<br>Не используется. |
| -901 | 335741018 | gfix_recon_fail | failed to reconnect to a transaction in database @1<br>Не используется. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -901 | 335741036 | gfix_trn_unknown | Transaction description item unknown<br>Не используется. |
| -901 | 335741038 | gfix_mode_req | "read_only" or "read_write" required<br>Не используется. |
| -901 | 335741042 | gfix_pzval_req | positive or zero numeric value required<br>Не используется. |
| -607 | 336003074 | dsql_dbkey_from_non_-table | Cannot SELECT RDB$DB_KEY from a stored procedure.<br>Невозможно выполнить SELECT RDB$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<br>Не используется. |
| 301 | 336003076 | dsql_dialect_warning_-expr | Use of @1 expression that returns different results in dialect 1 and dialect 3<br>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<br>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.<br>Не используется. |
| 301 | 336003080 | dsql_warning_number_-ambiguous | WARNING: Numeric literal @1 is interpreted as a floating-point<br>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<br>Двусмысленность имен полей между @1 и @2. |
| -607 | 336003086 | dsql_udf_return_pos_err | External function should have return position between 1 and @1<br>Внешняя функция должна иметь позицию возвращаемого значения между 1 и @1. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -104 | 336003087 | dsql_invalid_label | Label @1 @2 in the current scope<br>Метка @1 @2 находится в текущей области видимости. |
| -104 | 336003088 | dsql_datatypes_not_comparable | Datatypes @1 are not comparable in expression @2<br>Типы данных @1 не сравнимы в выражении @2. |
| -504 | 336003089 | dsql_cursor_invalid | Empty cursor name is not allowed<br>Имя курсора не может быть пустым. |
| -502 | 336003090 | dsql_cursor_redefined | Statement already has a cursor @1 assigned<br>У операторы уже установлен курсор @1. |
| -502 | 336003091 | dsql_cursor_not_found | Cursor @1 is not found in the current context<br>Курсор @1 не найден в текущем контексте. |
| -502 | 336003092 | dsql_cursor_exists | Cursor @1 already exists in the current context<br>Курсор @1 уже существует в текущем контексте. |
| -502 | 336003093 | dsql_cursor_rel_ambiguous | Relation @1 is ambiguous in cursor @2<br>Неоднозначная таблица @1 в курсоре @2. |
| -502 | 336003094 | dsql_cursor_rel_not_found | Relation @1 is not found in cursor @2<br>Таблица @1 не найдена в курсоре @2. |
| -502 | 336003095 | dsql_cursor_not_open | Cursor is not open<br>Курсор не открыт. |
| -607 | 336003096 | dsql_type_not_supp_ext_tab | Data type @1 is not supported for EXTERNAL TABLES. Relation '@2', field '@3'<br>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<br>Не используется. |
| -660 | 336003098 | primary_key_required | Primary key required on table @1<br>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<br>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<br>В UPDATE OR INSERT список полей не соответствует предложению MATCHING. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -817 | 336003101 | upd_ins_with_complex_-view | UPDATE OR INSERT without MATCHING could not be used with views based on more than one table<br>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<br>Несовместимый тип триггера. |
| -817 | 336003103 | dsql_db_trigger_type_-cant_change | Database trigger type can't be changed<br>Триггер на события базы данных не может быть изменен. |
| -607 | 336003104 | dsql_record_version_-table | To be used with RDB$RECORD_VERSION, @1 must be a table or a view of single table<br>Для использования с RDB$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<br>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<br>Пустой указатель на переменную индикатора NULL |
| -802 | 336003108 | dsql_no_sqldata | empty pointer to data<br>Пустой указатель на данные. |
| -802 | 336003109 | dsql_no_input_sqlda | No SQLDA for input values provided<br>Не указана SQLDA для входных значений. |
| -802 | 336003110 | dsql_no_output_sqlda | No SQLDA for output values provided<br>Не указана SQLDA для выходных значений. |
| -313 | 336003111 | dsql_wrong_param_num | Wrong number of parameters (expected @1, got @2)<br>Неверное количество параметров (ожидается @1, получено @2). |
| -817 | 336004072 | dsql_invalid_drop_ss_-clause | Invalid DROP SQL SECURITY clause<br>Недействительное условие DROP SQL SECURITY. |
| -901 | 336068645 | dyn_filter_not_found | BLOB Filter @1 not found<br>Не найден BLOB фильтр @1. |
| -901 | 336068649 | dyn_func_not_found | Function @1 not found<br>Не найдена функция @1. |
| -901 | 336068656 | dyn_index_not_found | Index not found<br>Индекс не найден. |
| -901 | 336068662 | dyn_view_not_found | View @1 not found<br>Представление @1 не найдено. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -901 | 336068697 | dyn_domain_not_found | Domain not found<br>Домен не найден. |
| -901 | 336068717 | dyn_cant_modify_auto_-trig | Triggers created automatically cannot be modified<br>Триггеры, созданные автоматически, не могут быть изменены. |
| -901 | 336068740 | dyn_dup_table | Table @1 already exists<br>Не используется. |
| -901 | 336068748 | dyn_proc_not_found | Procedure @1 not found<br>Процедура @1 не найдена. |
| -901 | 336068752 | dyn_exception_not_found | Exception not found<br>Исключение не найдено. |
| -901 | 336068754 | dyn_proc_param_not_-found | Parameter @1 in procedure @2 not found<br>Параметр @1 в процедуре @2 не найден. |
| -901 | 336068755 | dyn_trig_not_found | Trigger @1 not found<br>Триггер @1 не найден. |
| -901 | 336068759 | dyn_charset_not_found | Character set @1 not found<br>Набор символов @1 не найден. |
| -901 | 336068760 | dyn_collation_not_found | Collation @1 not found<br>Сортировка @1 не найдена. |
| -901 | 336068763 | dyn_role_not_found | Role @1 not found<br>Роль @1 не найдена. |
| -901 | 336068767 | dyn_name_longer | Name longer than database column size<br>Имя длиннее, чем размер столбца базы данных. |
| -901 | 336068784 | dyn_column_does_not_-exist | column @1 does not exist in table/view @2<br>Столбца @1 не существует в таблице/представлении @2. |
| -901 | 336068796 | dyn_role_does_not_exist | SQL role @1 does not exist<br>Не используется. |
| -901 | 336068797 | dyn_no_grant_admin_opt | user @1 has no grant admin option on SQL role @2<br>Не используется. |
| -901 | 336068798 | dyn_user_not_role_-member | user @1 is not a member of SQL role @2<br>Не используется. |
| -901 | 336068799 | dyn_delete_role_failed | @1 is not the owner of SQL role @2<br>Не используется. |
| -901 | 336068800 | dyn_grant_role_to_user | @1 is a SQL role and not a user<br>Не используется. |
| -901 | 336068801 | dyn_inv_sql_role_name | user name @1 could not be used for SQL role<br>Не используется. |
| -901 | 336068802 | dyn_dup_sql_role | SQL role @1 already exists<br>Не используется. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -901 | 336068803 | dyn_kywd_spec_for_role | keyword @1 can not be used as a SQL role name<br>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.<br>Не используется. |
| -612 | 336068812 | dyn_domain_name_exists | Cannot rename domain @1 to @2. A domain with that name already exists.<br>Не используется. |
| -612 | 336068813 | dyn_field_name_exists | Cannot rename column @1 to @2. A column with that name already exists in table @3.<br>Не используется. |
| -383 | 336068814 | dyn_dependency_exists | Column @1 from table @2 is referenced in @3<br>Не используется. |
| -315 | 336068815 | dyn_dtype_invalid | Cannot change datatype for column @1. Changing datatype is not supported for BLOB or ARRAY columns.<br>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.<br>Новый размер, указанный для столбца @1, должен быть по крайней мере @2 символов. |
| -829 | 336068817 | dyn_invalid_dtype_-conversion | Cannot change datatype for @1. Conversion from base type @2 to @3 is not supported.<br>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.<br>Невозможно изменить тип данных для столбца @1 из символьного типа в несимвольный. |
| -901 | 336068820 | dyn_zero_len_id | Zero length identifiers are not allowed<br>Идентификаторы с нулевой длиной не допускаются. |
| -901 | 336068822 | dyn_gen_not_found | Sequence @1 not found<br>Последовательность @1 не найдена. |
| -829 | 336068829 | max_coll_per_charset | Maximum number of collations per character set exceeded<br>В наборе символов превышено максимальное число сортировок. |
| -829 | 336068830 | invalid_coll_attr | Invalid collation attributes<br>Не используется. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -901 | 336068840 | dyn_wrong_gtt_scope | @1 cannot reference @2<br>Не используется. |
| -901 | 336068843 | dyn_coll_used_table | Collation @1 is used in table @2 (field name @3) and cannot be dropped<br>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<br>Сортировка @1 используется в домене @2 и не может быть удалена. |
| -607 | 336068845 | dyn_cannot_del_syscoll | Cannot delete system collation<br>Невозможно удалить системную сортировку. |
| -901 | 336068846 | dyn_cannot_del_def_coll | Cannot delete default collation of CHARACTER SET @1<br>Невозможно удалить сортировку по умолчанию для набора символов @1. |
| -901 | 336068849 | dyn_table_not_found | Table @1 not found<br>Таблица @1 не найдена. |
| -901 | 336068851 | dyn_coll_used_procedure | Collation @1 is used in procedure @2 (parameter name @3) and cannot be dropped<br>Сортировка @1 используется в процедуре @2 (имя параметра @3) и не может быть удалена. |
| -829 | 336068852 | dyn_scale_too_big | New scale specified for column @1 must be at most @2.<br>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.<br>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<br>Не используется. |
| -829 | 336068857 | dyn_cannot_addrem_-computed | Cannot add or remove COMPUTED from column @1<br>Не используется. |
| -901 | 336068858 | dyn_no_empty_pw | Password should not be empty string<br>Не используется. |
| -901 | 336068859 | dyn_dup_index | Index @1 already exists<br>Не используется. |
| -901 | 336068864 | dyn_package_not_found | Package @1 not found<br>Пакет @1 не найден. |
| -901 | 336068865 | dyn_schema_not_found | Schema @1 not found<br>Схема @1 не найдена. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -607 | 336068866 | dyn_cannot_mod_sysproc | Cannot ALTER or DROP system procedure @1<br>Невозможно изменить или удалить системную процедуру @1. |
| -607 | 336068867 | dyn_cannot_mod_systrig | Cannot ALTER or DROP system trigger @1<br>Невозможно изменить или удалить системный триггер @1. |
| -607 | 336068868 | dyn_cannot_mod_sysfunc | Cannot ALTER or DROP system function @1<br>Невозможно изменить или удалить системную функцию @1. |
| -607 | 336068869 | dyn_invalid_ddl_proc | Invalid DDL statement for procedure @1<br>Недопустимый оператор DDL для процедуры @1. |
| -607 | 336068870 | dyn_invalid_ddl_trig | Invalid DDL statement for trigger @1<br>Недопустимый оператор DDL для триггера @1. |
| -901 | 336068871 | dyn_funcnotdef_package | Function @1 has not been defined on the package body @2<br>Функция @1 не определена в теле пакета @2. |
| -901 | 336068872 | dyn_procnotdef_package | Procedure @1 has not been defined on the package body @2<br>Процедура @1 не определена в теле пакета @2. |
| -901 | 336068873 | dyn_funcsignat_package | Function @1 has a signature mismatch on package body @2<br>Сигнатура функции @1 не соответствует сигнатуре в теле пакета @2. |
| -901 | 336068874 | dyn_procsignat_package | Procedure @1 has a signature mismatch on package body @2<br>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<br>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<br>Тело пакета @1 уже существует. |
| -607 | 336068878 | dyn_invalid_ddl_func | Invalid DDL statement for function @1<br>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.<br>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<br>Параметр @1 в функции @2 не найден. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -901 | 336068887 | dyn_routine_param_not_found | Parameter @1 of routine @2 not found<br>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.<br>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<br>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<br>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<br>Не используется. |
| -901 | 336068894 | dyn_duplicate_package_item | Duplicate @1 @2<br>Дублирование @1 @2. |
| -901 | 336068895 | dyn_cant_modify_sysobj | System @1 @2 cannot be modified<br>Системный объект @1 @2 не может быть изменен. |
| -901 | 336068896 | dyn_cant_use_zero_increment | INCREMENT BY 0 is an illegal option for sequence @1<br>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<br>Не используется. |
| -901 | 336068898 | dyn_defvaldecl_package_func | Default values for parameters are allowed only in declaration of packaged function @1.@2<br>Значения по умолчанию для параметров разрешены только при объявлении функции в пакете @1.@2 |
| -901 | 336330753 | gbak_unknown_switch | found unknown switch<br>Не используется. |
| -901 | 336330754 | gbak_page_size_missing | page size parameter missing<br>Не используется. |
| -901 | 336330755 | gbak_page_size_toobig | Page size specified (@1) greater than limit (16384 bytes)<br>Не используется. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -901 | 336330756 | gbak_redir_ouput_-missing | redirect location for output is not specified<br>Не используется. |
| -901 | 336330757 | gbak_switches_conflict | conflicting switches for backup/restore<br>Не используется. |
| -901 | 336330758 | gbak_unknown_device | device type @1 not known<br>Не используется. |
| -901 | 336330759 | gbak_no_protection | protection is not there yet<br>Не используется. |
| -901 | 336330760 | gbak_page_size_not_-allowed | page size is allowed only on restore or create<br>Не используется. |
| -901 | 336330761 | gbak_multi_source_dest | multiple sources or destinations specified<br>Не используется. |
| -901 | 336330762 | gbak_filename_missing | requires both input and output filenames<br>Не используется. |
| -901 | 336330763 | gbak_dup_inout_names | input and output have the same name. Disallowed.<br>Не используется. |
| -901 | 336330764 | gbak_inv_page_size | expected page size, encountered "@1"<br>Не используется. |
| -901 | 336330765 | gbak_db_specified | REPLACE specified, but the first file @1 is a database<br>Не используется. |
| -901 | 336330766 | gbak_db_exists | database @1 already exists. To replace it, use the -REP switch<br>Не используется. |
| -901 | 336330767 | gbak_unk_device | device type not specified<br>Не используется. |
| -901 | 336330772 | gbak_blob_info_failed | gds_$blob_info failed<br>Не используется. |
| -901 | 336330773 | gbak_unk_blob_item | do not understand BLOB INFO item @1<br>Не используется. |
| -901 | 336330774 | gbak_get_seg_failed | gds_$get_segment failed<br>Не используется. |
| -901 | 336330775 | gbak_close_blob_failed | gds_$close_blob failed<br>Не используется. |
| -901 | 336330776 | gbak_open_blob_failed | gds_$open_blob failed<br>Не используется. |
| -901 | 336330777 | gbak_put_blr_gen_id_-failed | Failed in put_blr_gen_id<br>Ошибка в put_blr_gen_id. |
| -901 | 336330778 | gbak_unk_type | data type @1 not understood<br>Не используется. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -901 | 336330779 | gbak_comp_req_failed | gds_$compile_request failed<br>Не используется. |
| -901 | 336330780 | gbak_start_req_failed | gds_$start_request failed<br>Не используется. |
| -901 | 336330781 | gbak_rec_failed | gds_$receive failed<br>Не используется. |
| -901 | 336330782 | gbak_rel_req_failed | gds_$release_request failed<br>Не используется. |
| -901 | 336330783 | gbak_db_info_failed | gds_$database_info failed<br>Не используется. |
| -901 | 336330784 | gbak_no_db_desc | Expected database description record<br>Не используется. |
| -901 | 336330785 | gbak_db_create_failed | failed to create database @1<br>Не используется. |
| -901 | 336330786 | gbak_decomp_len_error | RESTORE: decompression length error<br>Не используется. |
| -901 | 336330787 | gbak_tbl_missing | cannot find table @1<br>Не используется. |
| -901 | 336330788 | gbak_blob_col_missing | Cannot find column for BLOB<br>Не используется. |
| -901 | 336330789 | gbak_create_blob_failed | gds_$create_blob failed<br>Не используется. |
| -901 | 336330790 | gbak_put_seg_failed | gds_$put_segment failed<br>Не используется. |
| -901 | 336330791 | gbak_rec_len_exp | expected record length<br>Не используется. |
| -901 | 336330792 | gbak_inv_rec_len | wrong length record, expected @1<br>encountered @2<br>Не используется. |
| -901 | 336330793 | gbak_exp_data_type | expected data attribute<br>Не используется. |
| -901 | 336330794 | gbak_gen_id_failed | Failed in store_blr_gen_id<br>Не используется. |
| -901 | 336330795 | gbak_unk_rec_type | do not recognize record type @1<br>Не используется. |
| -901 | 336330796 | gbak_inv_bkup_ver | Expected backup version 1..10. Found @1<br>Не используется. |
| -901 | 336330797 | gbak_missing_bkup_desc | expected backup description record<br>Не используется. |
| -901 | 336330798 | gbak_string_trunc | string truncated<br>Не используется. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -901 | 336330799 | gbak_cant_rest_record | warning - record could not be restored<br>Не используется. |
| -901 | 336330800 | gbak_send_failed | gds_$send failed<br>Не используется. |
| -901 | 336330801 | gbak_no_tbl_name | no table name for data<br>Не используется. |
| -901 | 336330802 | gbak_unexp_eof | unexpected end of file on backup file<br>Не используется. |
| -901 | 336330803 | gbak_db_format_too_old | database format @1 is too old to restore to<br>Не используется. |
| -901 | 336330804 | gbak_inv_array_dim | array dimension for column @1 is invalid<br>Не используется. |
| -901 | 336330807 | gbak_xdr_len_expected | Expected XDR record length<br>Не используется. |
| -901 | 336330817 | gbak_open_bkup_error | cannot open backup file @1<br>Не используется. |
| -901 | 336330818 | gbak_open_error | cannot open status and error output file @1<br>Не используется. |
| -901 | 336330934 | gbak_missing_block_fac | blocking factor parameter missing<br>Не используется. |
| -901 | 336330935 | gbak_inv_block_fac | expected blocking factor, encountered "@1"<br>Не используется. |
| -901 | 336330936 | gbak_block_fac_-specified | a blocking factor may not be used in conjunction with device CT<br>Не используется. |
| -901 | 336330940 | gbak_missing_username | user name parameter missing<br>Не используется. |
| -901 | 336330941 | gbak_missing_password | password parameter missing<br>Не используется. |
| -901 | 336330952 | gbak_missing_skipped_-bytes | missing parameter for the number of bytes to be skipped<br>Не используется. |
| -901 | 336330953 | gbak_inv_skipped_bytes | expected number of bytes to be skipped, encountered "@1"<br>Не используется. |
| -901 | 336330965 | gbak_err_restore_-charset | character set<br>Не используется. |
| -901 | 336330967 | gbak_err_restore_-collation | collation<br>Не используется. |
| -901 | 336330972 | gbak_read_error | Unexpected I/O error while reading from backup file<br>Не используется. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -901 | 336330973 | gbak_write_error | Unexpected I/O error while writing to backup file<br>Не используется. |
| -901 | 336330985 | gbak_db_in_use | could not drop database @1 (database might be in use)<br>Не используется. |
| -901 | 336330990 | gbak_sysmemex | System memory exhausted<br>Не используется. |
| -901 | 336331002 | gbak_restore_role_-failed | SQL role<br>Не используется. |
| -901 | 336331005 | gbak_role_op_missing | SQL role parameter missing<br>Не используется. |
| -901 | 336331010 | gbak_page_buffers_-missing | page buffers parameter missing<br>Не используется. |
| -901 | 336331011 | gbak_page_buffers_-wrong_param | expected page buffers, encountered "@1"<br>Не используется. |
| -901 | 336331012 | gbak_page_buffers_-restore | page buffers is allowed only on restore or create<br>Не используется. |
| -901 | 336331014 | gbak_inv_size | size specification either missing or incorrect for file @1<br>Не используется. |
| -901 | 336331015 | gbak_file_outof_-sequence | file @1 out of sequence<br>Не используется. |
| -901 | 336331016 | gbak_join_file_missing | can't join - one of the files missing<br>Не используется. |
| -901 | 336331017 | gbak_stdin_not_supptd | standard input is not supported when using join operation<br>Не используется. |
| -901 | 336331018 | gbak_stdout_not_supptd | standard output is not supported when using split operation or in verbose mode<br>Не используется. |
| -901 | 336331019 | gbak_bkup_corrupt | backup file @1 might be corrupt<br>Не используется. |
| -901 | 336331020 | gbak_unk_db_file_spec | database file specification missing<br>Не используется. |
| -901 | 336331021 | gbak_hdr_write_failed | can't write a header record to file @1<br>Не используется. |
| -901 | 336331022 | gbak_disk_space_ex | free disk space exhausted<br>Не используется. |
| -901 | 336331023 | gbak_size_lt_min | file size given (@1) is less than minimum allowed (@2)<br>Не используется. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -901 | 336331025 | gbak_svc_name_missing | service name parameter missing<br>Не используется. |
| -901 | 336331026 | gbak_not_ownr | Cannot restore over current database, must be SYSDBA or owner of the existing database.<br>Не используется. |
| -901 | 336331031 | gbak_mode_req | "read_only" or "read_write" required<br>Не используется. |
| -901 | 336331033 | gbak_just_data | just data ignore all constraints etc.<br>Не используется. |
| -901 | 336331034 | gbak_data_only | restoring data only ignoring foreign key, unique, not null & other constraints<br>Не используется. |
| -901 | 336331078 | gbak_missing_interval | verbose interval value parameter missing<br>Не используется. |
| -901 | 336331079 | gbak_wrong_interval | verbose interval value cannot be smaller than @1<br>Не используется. |
| -901 | 336331081 | gbak_verify_verbint | verify (verbose) and verbint options are mutually exclusive<br>Не используется. |
| -901 | 336331082 | gbak_option_only_-restore | option -@1 is allowed only on restore or create<br>Не используется. |
| -901 | 336331083 | gbak_option_only_backup | option -@1 is allowed only on backup<br>Не используется. |
| -901 | 336331084 | gbak_option_conflict | options -@1 and -@2 are mutually exclusive<br>Не используется. |
| -901 | 336331085 | gbak_param_conflict | parameter for option -@1 was already specified with value "@2"<br>Не используется. |
| -901 | 336331086 | gbak_option_repeated | option -@1 was already specified<br>Не используется. |
| -901 | 336331091 | gbak_max_dbkey_-recursion | dependency depth greater than @1 for view @2<br>Не используется. |
| -901 | 336331092 | gbak_max_dbkey_length | value greater than @1 when calculating length of rdb$db_key for view @2<br>Не используется. |
| -901 | 336331093 | gbak_invalid_metadata | Invalid metadata detected. Use -FIX_FSS_METADATA option.<br>Обнаружены недопустимые метаданные. Используйте параметр -FIX_FSS_METADATA. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -901 | 336331094 | gbak_invalid_data | Invalid data detected. Use -FIX_FSS_DATA option.<br>Обнаружены недопустимые данные. Используйте параметр -FIX_FSS_DATA. |
| -901 | 336331096 | gbak_inv_bkup_ver2 | Expected backup version @2..@3. Found @1<br>Не используется. |
| -901 | 336331100 | gbak_db_format_too_old2 | database format @1 is too old to backup<br>Не используется. |
| -804 | 336397205 | dsql_too_old_ods | ODS versions before ODS@1 are not supported<br>Не используется. |
| -607 | 336397206 | dsql_table_not_found | Table @1 does not exist<br>Таблица @1 не существует. |
| -607 | 336397207 | dsql_view_not_found | View @1 does not exist<br>Представление @1 не существует. |
| -206 | 336397208 | dsql_line_col_error | At line @1, column @2<br>Ошибка в строке @1, столбец @2. |
| -206 | 336397209 | dsql_unknown_pos | At unknown line and column<br>В неизвестных строке и столбце. |
| -206 | 336397210 | dsql_no_dup_name | Column @1 cannot be repeated in @2 statement<br>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<br>Не используется. |
| -637 | 336397213 | dsql_implicit_domain_name | Implicit domain name @1 not allowed in user created domain<br>Неявное доменное имя @1 не разрешено для доменов, создаваемых пользователями. |
| -607 | 336397214 | dsql_only_can_subscript_array | scalar operator used on field @1 which is not an array<br>Скалярный оператор, используемый в поле @1, которое не является массивом. |
| -104 | 336397215 | dsql_max_sort_items | cannot sort on more than 255 items<br>Не удается сортировать более, чем 255 элементов. |
| -104 | 336397216 | dsql_max_group_items | cannot group on more than 255 items<br>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<br>Не используется. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -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<br>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<br>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<br>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<br>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<br>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<br>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<br>Внутренняя ошибка DSQL: позиция столбца вышла из диапазона в pass1_union_auto_cast. |
| -104 | 336397225 | dsql_cte_wrong_-reference | Recursive CTE member (@1) can refer itself only in FROM clause<br>Рекурсивная часть CTE (@1) может ссылаться сама на себя только в предложении FROM. |
| -104 | 336397226 | dsql_cte_cycle | CTE '@1' has cyclic dependencies<br>CTE '@1' имеет циклические зависимости. |
| -104 | 336397227 | dsql_cte_outer_join | Recursive member of CTE can't be member of an outer join<br>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<br>Рекурсивная часть CTE не может ссылаться на себя более одного раза. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -104 | 336397229 | dsql_cte_not_a_union | Recursive CTE (@1) must be an UNION<br>Рекурсивный CTE (@1) должен содержать UNION. |
| -104 | 336397230 | dsql_cte_nonrecurs_-after_recurs | CTE '@1' defined non-recursive member after recursive<br>В CTE '@1' определена не рекурсивная часть после рекурсии. |
| -104 | 336397231 | dsql_cte_wrong_clause | Recursive member of CTE '@1' has @2 clause<br>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<br>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'<br>Не рекурсивная часть отсутствует в CTE '@1'. |
| -104 | 336397234 | dsql_cte_nested_with | WITH clause can't be nested<br>Предложение WITH не может быть вложенным. |
| -104 | 336397235 | dsql_col_more_than_-once_using | column @1 appears more than once in USING clause<br>Столбец @1 появляется более одного раза в разделе USING. |
| -901 | 336397236 | dsql_unsupp_feature_-dialect | feature is not supported in dialect @1<br>Это свойство не поддерживается в диалекте @1 |
| -104 | 336397237 | dsql_cte_not_used | CTE "@1" is not used in query<br>CTE "@1" не используется в запросе. |
| -104 | 336397238 | dsql_col_more_than_-once_view | column @1 appears more than once in ALTER VIEW<br>Столбец @1 появляется более одного раза в ALTER VIEW. |
| -901 | 336397239 | dsql_unsupported_in_-auto_trans | @1 is not supported inside IN AUTONOMOUS TRANSACTION block<br>@1 не поддерживается внутри блока IN AUTONOMOUS TRANSACTION. |
| -833 | 336397240 | dsql_eval_unknode | Unknown node type @1 in dsql/GEN_expr<br>Не используется. |
| -833 | 336397241 | dsql_agg_wrongarg | Argument for @1 in dialect 1 must be string or numeric<br>Аргумент для @1 в диалекте 1 должен быть строковым или числовым. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -833 | 336397242 | dsql_agg2_wrongarg | Argument for @1 in dialect 3 must be numeric<br>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<br>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<br>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<br>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<br>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<br>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<br>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<br>Недопустимый тип данных для умножения в диалекте 1. |
| -833 | 336397250 | dsql_nostring_multip_dial3 | Strings cannot be multiplied in dialect 3<br>В диалекте 3 строки умножать нельзя. |
| -833 | 336397251 | dsql_invalid_type_multip_dial3 | Invalid data type for multiplication in dialect 3<br>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<br>Деление в диалекте 3 должно быть между числовыми типа данных. |
| -833 | 336397253 | dsql_nostring_div_dial3 | Strings cannot be divided in dialect 3<br>В диалекте 3 сроки делить нельзя. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -833 | 336397254 | dsql_invalid_type_div_-dial3 | Invalid data type for division in dialect 3<br>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<br>Нельзя ставить знак «минус» перед строками в диалекте 3 |
| -833 | 336397256 | dsql_invalid_type_neg | Invalid data type for negation (minus operator)<br>Невозможно поставить знак «минус» перед указанным типом данных. |
| -104 | 336397257 | dsql_max_distinct_items | Cannot have more than 255 items in DISTINCT list<br>Невозможно содержать более 255 элементов в списке DISTINCT. |
| -901 | 336397258 | dsql_alter_charset_-failed | ALTER CHARACTER SET @1 failed<br>Ошибка в операторе ALTER CHARACTER SET @1. |
| -901 | 336397259 | dsql_comment_on_failed | COMMENT ON @1 failed<br>Ошибка в операторе COMMENT ON @1. |
| -901 | 336397260 | dsql_create_func_failed | CREATE FUNCTION @1 failed<br>Ошибка в операторе CREATE FUNCTION @1. |
| -901 | 336397261 | dsql_alter_func_failed | ALTER FUNCTION @1 failed<br>Ошибка в операторе ALTER FUNCTION @1. |
| -901 | 336397262 | dsql_create_alter_-func_failed | CREATE OR ALTER FUNCTION @1 failed<br>Ошибка в операторе CREATE OR ALTER FUNCTION @1. |
| -901 | 336397263 | dsql_drop_func_failed | DROP FUNCTION @1 failed<br>Ошибка в операторе DROP FUNCTION @1. |
| -901 | 336397264 | dsql_recreate_func_-failed | RECREATE FUNCTION @1 failed<br>Ошибка в операторе RECREATE FUNCTION @1. |
| -901 | 336397265 | dsql_create_proc_failed | CREATE PROCEDURE @1 failed<br>Ошибка в операторе CREATE PROCEDURE @1. |
| -901 | 336397266 | dsql_alter_proc_failed | ALTER PROCEDURE @1 failed<br>Ошибка в операторе ALTER PROCEDURE @1. |
| -901 | 336397267 | dsql_create_alter_-proc_failed | CREATE OR ALTER PROCEDURE @1 failed<br>Ошибка в операторе CREATE OR ALTER PROCEDURE @1. |
| -901 | 336397268 | dsql_drop_proc_failed | DROP PROCEDURE @1 failed<br>Ошибка в операторе DROP PROCEDURE @1. |
| -901 | 336397269 | dsql_recreate_proc_-failed | RECREATE PROCEDURE @1 failed<br>Ошибка в операторе RECREATE PROCEDURE @1. |
| -901 | 336397270 | dsql_create_trigger_-failed | CREATE TRIGGER @1 failed<br>Ошибка в операторе CREATE TRIGGER @1. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -901 | 336397271 | dsql_alter_trigger_-failed | ALTER TRIGGER @1 failed<br>Ошибка в операторе ALTER TRIGGER @1. |
| -901 | 336397272 | dsql_create_alter_-trigger_failed | CREATE OR ALTER TRIGGER @1 failed<br>Ошибка в операторе CREATE OR ALTER TRIGGER @1. |
| -901 | 336397273 | dsql_drop_trigger_-failed | DROP TRIGGER @1 failed<br>Ошибка в операторе DROP TRIGGER @1. |
| -901 | 336397274 | dsql_recreate_trigger_-failed | RECREATE TRIGGER @1 failed<br>Ошибка в операторе RECREATE TRIGGER @1. |
| -901 | 336397275 | dsql_create_collation_-failed | CREATE COLLATION @1 failed<br>Ошибка в операторе CREATE COLLATION @1. |
| -901 | 336397276 | dsql_drop_collation_-failed | DROP COLLATION @1 failed<br>Ошибка в операторе DROP COLLATION @1. |
| -901 | 336397277 | dsql_create_domain_-failed | CREATE DOMAIN @1 failed<br>Ошибка в операторе CREATE DOMAIN @1. |
| -901 | 336397278 | dsql_alter_domain_-failed | ALTER DOMAIN @1 failed<br>Ошибка в операторе ALTER DOMAIN @1. |
| -901 | 336397279 | dsql_drop_domain_failed | DROP DOMAIN @1 failed<br>Ошибка в операторе DROP DOMAIN @1. |
| -901 | 336397280 | dsql_create_except_-failed | CREATE EXCEPTION @1 failed<br>Ошибка в операторе CREATE EXCEPTION @1. |
| -901 | 336397281 | dsql_alter_except_-failed | ALTER EXCEPTION @1 failed<br>Ошибка в операторе ALTER EXCEPTION @1. |
| -901 | 336397282 | dsql_create_alter_-except_failed | CREATE OR ALTER EXCEPTION @1 failed<br>Ошибка в операторе CREATE OR ALTER EXCEPTION @1. |
| -901 | 336397283 | dsql_recreate_except_-failed | RECREATE EXCEPTION @1 failed<br>Ошибка в операторе RECREATE EXCEPTION @1. |
| -901 | 336397284 | dsql_drop_except_failed | DROP EXCEPTION @1 failed<br>Ошибка в операторе DROP EXCEPTION @1. |
| -901 | 336397285 | dsql_create_sequence_-failed | CREATE SEQUENCE @1 failed<br>Ошибка в операторе CREATE SEQUENCE @1. |
| -901 | 336397286 | dsql_create_table_-failed | CREATE TABLE @1 failed<br>Ошибка в операторе CREATE TABLE @1. |
| -901 | 336397287 | dsql_alter_table_failed | ALTER TABLE @1 failed<br>Ошибка в операторе ALTER TABLE @1. |
| -901 | 336397288 | dsql_drop_table_failed | DROP TABLE @1 failed<br>Ошибка в операторе DROP TABLE @1. |
| -901 | 336397289 | dsql_recreate_table_-failed | RECREATE TABLE @1 failed<br>Ошибка в операторе RECREATE TABLE @1. |
| -901 | 336397290 | dsql_create_pack_failed | CREATE PACKAGE @1 failed<br>Ошибка в операторе CREATE PACKAGE @1. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -901 | 336397291 | dsql_alter_pack_failed | ALTER PACKAGE @1 failed<br>Ошибка в операторе ALTER PACKAGE @1. |
| -901 | 336397292 | dsql_create_alter_-pack_failed | CREATE OR ALTER PACKAGE @1 failed<br>Ошибка в операторе CREATE OR ALTER PACKAGE @1. |
| -901 | 336397293 | dsql_drop_pack_failed | DROP PACKAGE @1 failed<br>Ошибка в операторе DROP PACKAGE @1. |
| -901 | 336397294 | dsql_recreate_pack_-failed | RECREATE PACKAGE @1 failed<br>Ошибка в операторе RECREATE PACKAGE @1. |
| -901 | 336397295 | dsql_create_pack_body_-failed | CREATE PACKAGE BODY @1 failed<br>Ошибка в операторе CREATE PACKAGE BODY @1. |
| -901 | 336397296 | dsql_drop_pack_body_-failed | DROP PACKAGE BODY @1 failed<br>Ошибка в операторе DROP PACKAGE BODY @1. |
| -901 | 336397297 | dsql_recreate_pack_-body_failed | RECREATE PACKAGE BODY @1 failed<br>Ошибка в операторе RECREATE PACKAGE BODY @1. |
| -901 | 336397298 | dsql_create_view_failed | CREATE VIEW @1 failed<br>Ошибка в операторе CREATE VIEW @1. |
| -901 | 336397299 | dsql_alter_view_failed | ALTER VIEW @1 failed<br>Ошибка в операторе ALTER VIEW @1. |
| -901 | 336397300 | dsql_create_alter_view_-failed | CREATE OR ALTER VIEW @1 failed<br>Ошибка в операторе CREATE OR ALTER VIEW @1. |
| -901 | 336397301 | dsql_recreate_view_-failed | RECREATE VIEW @1 failed<br>Ошибка в операторе RECREATE VIEW @1. |
| -901 | 336397302 | dsql_drop_view_failed | DROP VIEW @1 failed<br>Не используется. |
| -901 | 336397303 | dsql_drop_sequence_-failed | DROP SEQUENCE @1 failed<br>Ошибка в операторе DROP SEQUENCE @1. |
| -901 | 336397304 | dsql_recreate_sequence_-failed | RECREATE SEQUENCE @1 failed<br>Ошибка в операторе RECREATE SEQUENCE @1. |
| -901 | 336397305 | dsql_drop_index_failed | DROP INDEX @1 failed<br>Ошибка в операторе DROP INDEX @1. |
| -901 | 336397306 | dsql_drop_filter_failed | DROP FILTER @1 failed<br>Ошибка в операторе DROP FILTER @1. |
| -901 | 336397307 | dsql_drop_shadow_failed | DROP SHADOW @1 failed<br>Ошибка в операторе DROP SHADOW @1. |
| -901 | 336397308 | dsql_drop_role_failed | DROP ROLE @1 failed<br>Ошибка в операторе DROP ROLE @1. |
| -901 | 336397309 | dsql_drop_user_failed | DROP USER @1 failed<br>Ошибка в операторе DROP USER @1. |
| -901 | 336397310 | dsql_create_role_failed | CREATE ROLE @1 failed<br>Ошибка в операторе CREATE ROLE @1. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -901 | 336397311 | dsql_alter_role_failed | ALTER ROLE @1 failed<br>Не используется. |
| -901 | 336397312 | dsql_alter_index_failed | ALTER INDEX @1 failed<br>Ошибка в операторе ALTER INDEX @1. |
| -901 | 336397313 | dsql_alter_database_-failed | ALTER DATABASE failed<br>Ошибка в операторе ALTER DATABASE @1. |
| -901 | 336397314 | dsql_create_shadow_-failed | CREATE SHADOW @1 failed<br>Ошибка в операторе CREATE SHADOW @1. |
| -901 | 336397315 | dsql_create_filter_-failed | DECLARE FILTER @1 failed<br>Ошибка в операторе DECLARE FILTER @1. |
| -901 | 336397316 | dsql_create_index_-failed | CREATE INDEX @1 failed<br>Ошибка в операторе CREATE INDEX @1. |
| -901 | 336397317 | dsql_create_user_failed | CREATE USER @1 failed<br>Ошибка в операторе CREATE USER @1. |
| -901 | 336397318 | dsql_alter_user_failed | ALTER USER @1 failed<br>Ошибка в операторе ALTER USER @1. |
| -901 | 336397319 | dsql_grant_failed | GRANT failed<br>Ошибка в операторе GRANT. |
| -901 | 336397320 | dsql_revoke_failed | REVOKE failed<br>Ошибка в операторе REVOKE. |
| -104 | 336397321 | dsql_cte_recursive_-aggregate | Recursive member of CTE cannot use aggregate or window function<br>Рекурсивный член CTE не может использовать агрегатные или оконные функции. |
| -901 | 336397322 | dsql_mapping_failed | @2 MAPPING @1 failed<br>Ошибка в операторе @2 MAPPING @1. |
| -901 | 336397323 | dsql_alter_sequence_-failed | ALTER SEQUENCE @1 failed<br>Ошибка в операторе ALTER SEQUENCE @1. |
| -901 | 336397324 | dsql_create_generator_-failed | CREATE GENERATOR @1 failed<br>Не используется. |
| -901 | 336397325 | dsql_set_generator_-failed | SET GENERATOR @1 failed<br>Ошибка в операторе SET GENERATOR @1. |
| -104 | 336397326 | dsql_wlock_simple | WITH LOCK can be used only with a single physical table<br>Предложение WITH LOCK доступно только для выборки данных из одной таблицы. |
| -104 | 336397327 | dsql_firstskip_rows | FIRST/SKIP cannot be used with OFFSET/FETCH or ROWS<br>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<br>Предложение WITH LOCK нельзя использовать при использовании любых агрегатных функций. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -104 | 336397329 | dsql_wlock_conflict | WITH LOCK cannot be used with @1<br>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<br>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<br>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<br>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<br>Слишком много вложенных блоков BEGIN...END, максимальный уровень - @1. |
| -901 | 336398288 | dsql_create_policy_-failed | CREATE POLICY @1 failed<br>Не используется. |
| -901 | 336398289 | dsql_alter_policy_-failed | ALTER POLICY @1 failed<br>Не используется. |
| -901 | 336398290 | dsql_drop_policy_failed | DROP POLICY @1 failed<br>Не используется. |
| -901 | 336723983 | gsec_cant_open_db | unable to open database<br>Не используется. |
| -901 | 336723984 | gsec_switches_error | error in switch specifications<br>Ошибка в задании переключателя. |
| -901 | 336723985 | gsec_no_op_spec | no operation specified<br>Не используется. |
| -901 | 336723986 | gsec_no_usr_name | no user name specified<br>Не используется. |
| -901 | 336723987 | gsec_err_add | add record error<br>Не используется. |
| -901 | 336723988 | gsec_err_modify | modify record error<br>Не используется. |
| -901 | 336723989 | gsec_err_find_mod | find/modify record error<br>Не используется. |
| -901 | 336723990 | gsec_err_rec_not_found | record not found for user: @1<br>Не используется. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -901 | 336723991 | gsec_err_delete | delete record error<br>Не используется. |
| -901 | 336723992 | gsec_err_find_del | find/delete record error<br>Не используется. |
| -901 | 336723996 | gsec_err_find_disp | find/display record error<br>Не используется. |
| -901 | 336723997 | gsec_inv_param | invalid parameter, no switch defined<br>Недопустимый параметр, такой переключатель не определен. |
| -901 | 336723998 | gsec_op_specified | operation already specified<br>Не используется. |
| -901 | 336723999 | gsec_pw_specified | password already specified<br>Не используется. |
| -901 | 336724000 | gsec_uid_specified | uid already specified<br>Не используется. |
| -901 | 336724001 | gsec_gid_specified | gid already specified<br>Не используется. |
| -901 | 336724002 | gsec_proj_specified | project already specified<br>Не используется. |
| -901 | 336724003 | gsec_org_specified | organization already specified<br>Не используется. |
| -901 | 336724004 | gsec_fname_specified | first name already specified<br>Не используется. |
| -901 | 336724005 | gsec_mname_specified | middle name already specified<br>Не используется. |
| -901 | 336724006 | gsec_lname_specified | last name already specified<br>Не используется. |
| -901 | 336724008 | gsec_inv_switch | invalid switch specified<br>Не используется. |
| -901 | 336724009 | gsec_amb_switch | ambiguous switch specified<br>Не используется. |
| -901 | 336724010 | gsec_no_op_specified | no operation specified for parameters<br>Не используется. |
| -901 | 336724011 | gsec_params_not_allowed | no parameters allowed for this operation<br>Параметры недопустимы для этой операции. |
| -901 | 336724012 | gsec_incompat_switch | incompatible switches specified<br>Не используется. |
| -901 | 336724044 | gsec_inv_username | Invalid user name (maximum 31 bytes allowed)<br>Не используется. |
| -901 | 336724045 | gsec_inv_pw_length | Warning - maximum 8 significant bytes of password used<br>Не используется. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -901 | 336724046 | gsec_db_specified | database already specified<br>Не используется. |
| -901 | 336724047 | gsec_db_admin_specified | database administrator name already specified<br>Не используется. |
| -901 | 336724048 | gsec_db_admin_pw_-specified | database administrator password already specified<br>Не используется. |
| -901 | 336724049 | gsec_sql_role_specified | SQL role name already specified<br>Не используется. |
| -901 | 336920577 | gstat_unknown_switch | found unknown switch<br>Не используется. |
| -901 | 336920578 | gstat_retry | please retry, giving a database name<br>Не используется. |
| -901 | 336920579 | gstat_wrong_ods | Wrong ODS version, expected @1, encountered @2<br>Не используется. |
| -901 | 336920580 | gstat_unexpected_eof | Unexpected end of database file.<br>Не используется. |
| -901 | 336920605 | gstat_open_err | Can't open database file @1<br>Не используется. |
| -901 | 336920606 | gstat_read_err | Can't read a database page<br>Не используется. |
| -901 | 336920607 | gstat_sysmemex | System memory exhausted<br>Не используется. |
| -901 | 336986113 | fbsvcmgr_bad_am | Wrong value for access mode<br>Неверное значение для режима доступа. |
| -901 | 336986114 | fbsvcmgr_bad_wm | Wrong value for write mode<br>Неверное значение для режима записи. |
| -901 | 336986115 | fbsvcmgr_bad_rs | Wrong value for reserve space<br>Неверное значение для зарезервированного пространства. |
| -901 | 336986116 | fbsvcmgr_info_err | Unknown tag (@1) in info_svr_db_info block after isc_svc_query()<br>Неизвестный тег (@1) в блоке info_svr_db_info после вызова isc_svc_query(). |
| -901 | 336986117 | fbsvcmgr_query_err | Unknown tag (@1) in isc_svc_query() results<br>Неизвестный тег (@1) в результатах isc_svc_query() |
| -901 | 336986118 | fbsvcmgr_switch_unknown | Unknown switch "@1"<br>Неизвестный параметр командной строки "@1". |
| -901 | 336986159 | fbsvcmgr_bad_sm | Wrong value for shutdown mode<br>Неверное значение для режима останова. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -901 | 336986160 | fbsvcmgr_fp_open | could not open file @1<br>Не удается открыть файл @1. |
| -901 | 336986161 | fbsvcmgr_fp_read | could not read file @1<br>Не удается прочитать файл @1. |
| -901 | 336986162 | fbsvcmgr_fp_empty | empty file @1<br>Пустой файл @1. |
| -901 | 336986164 | fbsvcmgr_bad_arg | Invalid or missing parameter for switch @1<br>Недопустимый или отсутствующий параметр для переключателя @1. |
| -901 | 337051649 | utl_trusted_switch | Switches trusted_user and trusted_role are not supported from command line<br>Переключатели trusted_user и trusted_role не поддерживаются в командной строке. |
| -901 | 337117213 | nbackup_missing_param | Missing parameter for switch @1<br>Отсутствует параметр для переключателя @1. |
| -901 | 337117214 | nbackup_allowed_-switches | Only one of -LOCK, -UNLOCK, -FIXUP, -BACKUP or -RESTORE should be specified<br>Необходимо указать только один из: -LOCK, -UNLOCK, -FIXUP, -BACKUP или -RESTORE. |
| -901 | 337117215 | nbackup_unknown_param | Unrecognized parameter @1<br>Нераспознанный параметр @1. |
| -901 | 337117216 | nbackup_unknown_switch | Unknown switch @1<br>Неизвестный переключатель @1. |
| -901 | 337117217 | nbackup_nofetchpw_svc | Fetch password can't be used in service mode<br>Исходный пароль нельзя использовать в сервисном режиме. |
| -901 | 337117218 | nbackup_pwfile_error | Error working with password file "@1"<br>Ошибка при работе с файлом пароля "@1". |
| -901 | 337117219 | nbackup_size_with_lock | Switch -SIZE can be used only with -LOCK<br>Переключатель -SIZE может быть использован только вместе в -LOCK. |
| -901 | 337117220 | nbackup_no_switch | None of -LOCK, -UNLOCK, -FIXUP, -BACKUP or -RESTORE specified<br>Не указана ни одна опция: -LOCK, -UNLOCK, -FIXUP, -BACKUP или -RESTORE. |
| -901 | 337117223 | nbackup_err_read | IO error reading file: @1<br>Ошибка ввода-вывода при чтении файла: @1. |
| -901 | 337117224 | nbackup_err_write | IO error writing file: @1<br>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<br>Ошибка открытия файла базы данных: @1. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -901 | 337117227 | nbackup_err_fadvice | Error in posix_fadvise(@1) for database @2<br>Ошибка в posix_fadvise (@1) для базы данных @ 2. |
| -901 | 337117228 | nbackup_err_createdb | Error creating database file: @1<br>Ошибка при создании файла базы данных: @1. |
| -901 | 337117229 | nbackup_err_openbk | Error opening backup file: @1<br>Ошибка при открытии файла бэкапа: @1. |
| -901 | 337117230 | nbackup_err_createbk | Error creating backup file: @1<br>Ошибка при создании файла бэкапа: @1. |
| -901 | 337117231 | nbackup_err_eofdb | Unexpected end of database file @1<br>Неожиданное окончание файла базы данных @1. |
| -901 | 337117232 | nbackup_fixup_-wrongstate | Database @1 is not in state (@2) to be safely fixed up<br>База данных @1 не в состоянии (@2), чтобы быть безопасно починенной. |
| -901 | 337117233 | nbackup_err_db | Database error<br>Ошибка базы данных. |
| -901 | 337117234 | nbackup_userpw_toolong | Username or password is too long<br>Логин или пароль слишком длинный. |
| -901 | 337117235 | nbackup_lostrec_db | Cannot find record for database "@1" backup level @2 in the backup history<br>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<br>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)<br>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<br>Внутренняя ошибка. Файл базы данных не заблокирован. Флаги @1. |
| -901 | 337117239 | nbackup_lostguid_bk | Internal error. Cannot get backup guid clumplet<br>Внутренняя ошибка. Не удается получить GUID резервной копии. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -901 | 337117240 | nbackup_page_changed | Internal error. Database page @1 had been changed during backup (page SCN=@2, backup SCN=@3)<br>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<br>Размер файла базы данных не кратен размеру страницы. |
| -901 | 337117242 | nbackup_failed_lzbk | Level 0 backup is not restored<br>Резервная копия уровня 0 не восстанавливается. |
| -901 | 337117243 | nbackup_err_eofhdrbk | Unexpected end of file when reading header of backup file: @1<br>Неожиданный конец файла при чтении заголовка файла резервной копии: @1. |
| -901 | 337117244 | nbackup_invalid_incbk | Invalid incremental backup file: @1<br>Недопустимый инкрементный файл резервной копии: @1. |
| -901 | 337117245 | nbackup_unsupvers_incbk | Unsupported version @1 of incremental backup file: @2<br>Неподдерживаемая версия @1 файла инкрементной резервной копии: @2. |
| -901 | 337117246 | nbackup_invlevel_incbk | Invalid level @1 of incremental backup file: @2, expected @3<br>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<br>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<br>Неожиданный конец файла резервной копии: @1. |
| -901 | 337117249 | nbackup_err_copy | Error creating database file: @1 via copying from: @2<br>Ошибка при создании файла базы данных: @1 путем копирования из: @2. |
| -901 | 337117250 | nbackup_err_eofhdr_-restdb | Unexpected end of file when reading header of restored database file (stage @1)<br>Неожиданный конец файла при чтении заголовка восстановленного файла базы данных (состояние @1). |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -901 | 337117251 | nbackup_lostguid_l0bk | Cannot get backup guid clumplet from L0 backup<br>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<br>Неверный параметр @1 для переключателя -D, нужен: ON или OFF. |
| -901 | 337117257 | nbackup_user_stop | Terminated due to user request<br>Завершение, обусловленное запросом пользователя. |
| -901 | 337117259 | nbackup_deco_parse | Too complex decompress command (> @1 arguments)<br>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<br>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<br>Найдены конфликтующие действия "@1" и "@2". |
| -901 | 337182751 | trace_act_notfound | action switch not found<br>Переключатель действия не найден. |
| -901 | 337182752 | trace_switch_once | switch "@1" must be set only once<br>Переключатель "@1" должен быть установлен только один раз. |
| -901 | 337182753 | trace_param_val_miss | value for switch "@1" is missing<br>Значение для переключателя "@1" отсутствует. |
| -901 | 337182754 | trace_param_invalid | invalid value ("@1") for switch "@2"<br>Неверное значение ("@1") переключателя "@2". |
| -901 | 337182755 | trace_switch_unknown | unknown switch "@1" encountered<br>Неизвестный переключатель "@1". |
| -901 | 337182756 | trace_switch_svc_only | switch "@1" can be used by service only<br>Не используется |
| -901 | 337182757 | trace_switch_user_only | switch "@1" can be used by interactive user only<br>Переключатель "@1" может использоваться только интерактивным пользователем. |
| -901 | 337182758 | trace_switch_param_miss | mandatory parameter "@1" for switch "@2" is missing<br>Обязательный параметр "@1" для переключателя "@2" отсутствует. |

| SQLCODE | GDSCODE | Символ | Текст сообщения |
|---|---|---|---|
| -901 | 337182759 | trace_param_act_-notcompat | parameter "@1" is incompatible with action "@2"<br>Параметр "@1" несовместим с действием "@2". |
| -901 | 337182760 | trace_mandatory_switch_-miss | mandatory switch "@1" is missing<br>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 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 — Коды ошибок SQLSTATE

| Код SQLSTATE | Текст сообщения | Описание |
|---|---|---|
| **SQLCLASS 00 (Success)** | | |
| 00000 | Success | Успешное выполнение |
| **SQLCLASS 01 (Warning)** | | |
| 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 |
| **SQLCLASS 02 (No Data)** | | |
| 02000 | No data found or no rows affected | Не найдено ни данных, ни строк |

| Код SQLSTATE | Текст сообщения | Описание |
|---|---|---|
| **SQLCLASS 07 (Dynamic SQL error)** | | |
| 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 |
| 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 |
| **SQLCLASS 08 (Connection Exception)** | | |
| 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 |
| **SQLCLASS 0A (Feature Not Supported)** | | |
| 0A000 | Feature not supported | Возможность не поддерживается |
| **SQLCLASS 0B (Invalid Transaction Initiation)** | | |
| 0B000 | Invalid transaction initiation | Неверная инициализация транзакции |
| **SQLCLASS 0L (Invalid Grantor)** | | |
| 0L000 | Invalid grantor | Неверный грантор (тот, кто дает привилегии) |
| **SQLCLASS 0P (Invalid Role Specification)** | | |
| 0P000 | Invalid role specification | Неверная спецификация роли |
| **SQLCLASS 0U (Attempt to Assign to Non-Updatable Column)** | | |
| 0U000 | Attempt to assign to non-updatable column | Попытка присвоения не обновляемому столбцу |
| **SQLCLASS 0V (Attempt to Assign to Ordering Column)** | | |
| 0V000 | Attempt to assign to Ordering Column | Попытка присвоения сортируемому столбцу |
| **SQLCLASS 20 (Case Not Found For Case Statement)** | | |

| Код SQLSTATE | Текст сообщения | Описание |
|---|---|---|
| 20000 | Case not found for case statement | Не обнаружено вариантов для предложения CASE |
| **SQLCLASS 21 (Cardinality Violation)** | | |
| 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 |
| **SQLCLASS 22 (Data Exception)** | | |
| 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 |
| 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 |

| Код SQLSTATE | Текст сообщения | Описание |
|---|---|---|
| 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 |
| 2202E | Array element error | Ошибка элемента массива |
| 2202F | Array data, right truncation | Данные массива, обрезание справа |
| **SQLCLASS 23 (Integrity Constraint Violation)** | | |
| 23000 | Integrity constraint violation | Нарушение ограничения целостности |
| **SQLCLASS 24 (Invalid Cursor State)** | | |
| 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 |
| **SQLCLASS 25 (Invalid Transaction State)** | | |
| 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 |
| **SQLCLASS 26 (Invalid SQL Statement Name)** | | |
| 26000 | Invalid SQL statement name | Неверное имя SQL выражения |
| **SQLCLASS 27 (Triggered Data Change Violation)** | | |
| 27000 | Triggered data change violation | Ошибки при изменении данных триггером |
| **SQLCLASS 28 (Invalid Authorization Specification)** | | |
| 28000 | Invalid authorization specification | Неверная спецификация авторизации |
| **SQLCLASS 2B (Dependent Privilege Descriptors Still Exist)** | | |

| Код SQLSTATE | Текст сообщения | Описание |
|---|---|---|
| 2B000 | Dependent privilege descriptors still exist | Зависимые описания привилегий еще существуют |
| **SQLCLASS 2C (Invalid Character Set Name)** | | |
| 2C000 | Invalid character set name | Неверное имя набора символов |
| **SQLCLASS 2D (Invalid Transaction Termination)** | | |
| 2D000 | Invalid transaction termination | Неверное завершение транзакции |
| **SQLCLASS 2E (Invalid Connection Name)** | | |
| 2E000 | Invalid connection name | Неверное имя соединения |
| **SQLCLASS 2F (SQL Routine Exception)** | | |
| 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 |
| **SQLCLASS 33 (Invalid SQL Descriptor Name)** | | |
| 33000 | Invalid SQL descriptor name | Недопустимое имя дескриптора SQL |
| **SQLCLASS 34 (Invalid Cursor Name)** | | |
| 34000 | Invalid cursor name | Неверное имя курсора |
| **SQLCLASS 35 (Invalid condition number)** | | |
| 35000 | Invalid condition number | Неверный номер условия |
| **SQLCLASS 36 (Cursor Sensitivity Exception)** | | |
| 36001 | Request rejected | Запрос отказан |
| 36002 | Request failed | Запрос ошибочный |
| **SQLCLASS 38 (External Routine Exception)** | | |
| 38000 | External routine exception | Ошибка внешней процедуры |
| **SQLCLASS 39 (External Routine Invocation Exception)** | | |
| 39000 | External routine invocation exception | Ошибка вызова внешней процедуры |
| **SQLCLASS 3B (Invalid Save Point)** | | |
| 3B000 | Invalid save point | Неверная точка сохранения |
| **SQLCLASS 3C (Ambiguous Cursor Name)** | | |
| 3C000 | Ambiguous cursor name | Имя курсора неоднозначное |
| **SQLCLASS 3D (Invalid Catalog Name)** | | |
| 3D000 | Invalid catalog name | Неверное имя каталога |
| 3D001 | Catalog name not found | Каталог с таким именем не обнаружен |
| **SQLCLASS 3F (Invalid Schema Name)** | | |

| Код SQLSTATE | Текст сообщения | Описание |
|---|---|---|
| 3F000 | Invalid schema name | Неверное имя схемы |
| **SQLCLASS 40 (Transaction Rollback)** | | |
| 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 |
| **SQLCLASS 42 (Syntax Error or Access Violation)** | | |
| 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 |
| **SQLCLASS 44 (With Check Option Violation)** | | |
| 44000 | WITH CHECK OPTION violation | Нарушение опции WITH CHECK |
| **SQLCLASS 45 (Unhandled user-defined exception)** | | |
| 45000 | Unhandled user-defined exception | Необработанное исключение, определенное пользователем |
| **SQLCLASS 54 (Program Limit Exceeded)** | | |
| 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 |
| **SQLCLASS HY (CLI-specific condition)** | | |
| 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 |

| Код SQLSTATE | Текст сообщения | Описание |
|---|---|---|
| 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 |
| 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 |

| Код SQLSTATE | Текст сообщения | Описание |
|---|---|---|
| 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 |
| 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 |
| **SQLCLASS XX (Internal Error)** | | |
| XX000 | Internal error | Внутренняя ошибка |
| XX001 | Data corrupted | Данные разрушены |
| XX002 | Index corrupted | Индекс разрушен |

# Приложение B Наборы символов и порядки сортировки

Наборы символов в Ред База Данных и соответствующие им порядки сортировки представлены в таблице B.1.

Таблица B.1 — Наборы символов и порядки сортировки Ред База Данных

| ID | Название | Байтов на символ | Порядок сортировки | Язык |
|---|---|---|---|---|
| 2 | ASCII | 1 | ASCII | Английский |
| 56 | BIG_5 | 2 | BIG_5 | Китайский,<br>Вьетнамский,<br>Корейский |
| 50 | CYRL | 1 | CYRL<br>DB_RUS<br>PDOX_CYRL | Русский,<br>Русский dBase,<br>Русский Paradox |
| 10 | DOS437 | 1 | DOS437<br>DB_DEU437<br>DB_ESP437<br>DB_FIN437<br>DB_FRA437<br>DB_ITA437<br>DB_NLD437<br>DB_SVE437<br>DB_UK437<br>DB_US437<br>PDOX_ASCII<br>PDOX_SWEDFIN<br>PDOX_INTL | Английский—США,<br>Немецкий dBase,<br>Испанский dBase,<br>Финский dBase,<br>Французский dBase,<br>Итальянский dBase,<br>Голландский dBase,<br>Шведский dBase,<br>Английский (Великобритания) dBase,<br>Английский (США) dBase,<br>Кодовая страница Paradox—ASCII,<br>Paradox Шведская / Финская кодовые страницы,<br>Paradox международный английский кодовая страница |
| 9 | DOS737 | 1 | DOS737 | Греческий |
| 15 | DOS775 | 1 | DOS775 | Балтийский |

| ID | Название | Байтов на символ | Порядок сортировки | Язык |
|---|---|---|---|---|
| 11 | DOS850 | 1 | DOS850<br>DB_DEU850<br>DB_ESP850<br>DB_FRA850<br>DB_FRC850<br>DB_ITA850<br>DB_NLD850<br>DB_PTB850<br>DB_SVE850<br>DB_UK850<br>DB_US850 | Латинский I (нет символа Евро),<br>Немецкий,<br>Испанский,<br>Французский<br>Французский — Канада,<br>Итальянский,<br>Голландский,<br>Португальский — Бразилия,<br>Шведский,<br>Английский — Великобритания,<br>Английский — США. |
| 45 | DOS852 | 1 | DOS852<br>DB_CSY<br>DB_PLK<br>DB_SLO<br>PDOX_CSY<br>PDOX_HUN<br>PDOX_PLK<br>PDOX_SLO | Латинский II,<br>Чешский dBase,<br>Польский dBase,<br>Словацкий dBase,<br>Чешский Paradox,<br>Венгерский Paradox,<br>Польский Paradox,<br>Словацкий Paradox. |
| 46 | DOS857 | 1 | DOS857<br>DB_TRK | Турецкий,<br>Турецкий dBase. |
| 16 | DOS858 | 1 | DOS858 | Латинский I с символом Евро. |
| 13 | DOS860 | 1 | DOS860<br>DB_PTG860 | Португальский,<br>Португальский dBase. |
| 47 | DOS861 | 1 | DOS861<br>PDOX_ISL | Исландский,<br>Исландский Paradox. |
| 17 | DOS862 | 1 | DOS862 | Иврит |
| 14 | DOS863 | 1 | DOS863<br>DB_FRC863 | Французский — Канада,<br>Французский dBase — Канада |
| 18 | DOS864 | 1 | DOS864 | Арабский |
| 12 | DOS865 | 1 | DOS865<br>DB_DAN865<br>DB_NOR865<br>PDOX_NORDAN4 | Скандинавские,<br>Датский dBase,<br>Норвежский dBase,<br>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) |

| ID | Название | Байтов на символ | Порядок сортировки | Язык |
|---|---|---|---|---|
| 21 | ISO8859_1 | 1 | ISO8859_1<br>DA_DA<br>DE_DE<br>DU_NL<br>EN_UK<br>EN_US<br>ES_ES<br>ES_ES_CI_AI<br>FI_FI<br>FR_CA<br>FR_FR<br>IS_IS<br>IT_IT<br>NO_NO<br>PT_PT<br>PT_BR<br>SV_SV | Латинский I,<br>Датский,<br>Немецкий,<br>Голландский,<br>Английский, Великобритания,<br>Английский, США,<br>Испанский,<br>Испанский без различия строчных и прописных букв и без знаков ударения<br>Финский,<br>Французский, Канада,<br>Французский,<br>Исландский,<br>Итальянский,<br>Норвежский,<br>Португальский,<br>Португальский, Бразилия.<br>Шведский. |
| 22 | ISO8859_2 | 1 | ISO8859_2<br>CS_CZ<br>ISO_HUN<br>ISO_PLK | Латинский 2 — Центральная Европа (хорватский, чешский, венгерский, польский, румынский, сербский, словацкий, словенский),<br>Чешский,<br>Венгерский,<br>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<br>LT_LT | Латинский 7 — Балтика<br>Литовский |
| 63 | KOI8R | 1 | KOI8R<br>KOI8R_RU | Русский. Словарное упорядочение.<br>Русский |
| 64 | KOI8U | 1 | KOI8U<br>KOI8R_UA | Украинский. Словарное упорядочение.<br>Украинский |
| 44 | KSC_5601 | 2 | KSC_5601<br>KSC_DICTIONARY | Корейский,<br>Корейский — словарный порядок сортировки |

| ID | Название | Байтов на символ | Порядок сортировки | Язык |
|---|---|---|---|---|
| 19 | NEXT | 1 | NEXT<br>NXT_DEU<br>NXT_ESP<br>NXT_FRA<br>NXT_ITA<br>NXT_US | Кодирование NeXTSTEP,<br>Немецкий,<br>Испанский,<br>Французский,<br>Итальянский,<br>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. |
| 5 | SJIS_0208 | 2 | SJIS_0208 | Японский. |
| 3 | UNICODE_FSS | 3 | UNICODE_FSS | UNICODE |
| 4 | UTF8 | 4 | UTF8<br>USC_BASIC<br>UNICODE | UNICODE 4.0.<br>UNICODE 4.0.<br>UNICODE 4.0. |
| 51 | WIN1250 | 1 | WIN1250<br>BS_BA<br>PXW_CSY<br>PXW_HUN<br>PXW_HUNDC<br>PXW_PLK<br>PXW_SLOV<br>WIN_CZ<br>WIN_CZ_CI_AI | ANSI — Центральная Европа,<br>Боснийский.<br>Венгерский,<br>Венгерский — словарная сортировка,<br>Польский,<br>Словацкий,<br>Словенский,<br>Чешский без различия строчных и прописных букв.<br>Чешский без различия строчных и прописных букв, нечувствительный к знакам ударения. |
| 52 | WIN1251 | 1 | WIN1251<br>WIN1251_UA<br>PXW_CYRL | ANSI кириллица,<br>Украинский,<br>Paradox кириллица (русский) |
| 53 | WIN1252 | 1 | WIN1252<br>PXW_INTL<br>PXW_INTL850<br>PXW_NORDAN4<br>PXW_SPAN<br>PXW_SWEDFIN<br>WIN_PTBR | ANSI — Латинский I,<br>Английский интернациональный,<br>Paradox многоязыковый Латинский I,<br>Норвежский и датский,<br>Paradox испанский,<br>Шведский и финский,<br>Португальский, Бразилия |
| 54 | WIN1253 | 1 | WIN1253<br>PXW_GREEK | ANSI греческий,<br>Paradox греческий. |
| 55 | WIN1254 | 1 | WIN1254<br>PXW_TURK | ANSI турецкий,<br>Paradox турецкий. |

| ID | Название | Байтов на символ | Порядок сортировки | Язык |
|---|---|---|---|---|
| 58 | WIN1255 | 1 | WIN1255 | ANSI иврит. |
| 59 | WIN1256 | 1 | WIN1256 | ANSI арабский. |
| 60 | WIN1257 | 1 | WIN1257<br>WIN1257_EE<br>WIN1257_LT<br>WIN1257_LV | ANSI балтийский,<br>Эстонский. Словарное упорядочение.<br>Литовский. Словарное упорядочение.<br>Латвийский. Словарное упорядочение. |
| 65 | WIN1258 | 1 | WIN1258 | Вьетнамский |

# Приложение Г 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.

## 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 Г.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.

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.

## 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 Г.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.

## 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 Г.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.

В комплект поставки Ред База Данных входят две библиотеки, содержащие функции, определенные пользователем — ib_udf.dll и fbudf.dll.

Вместе с функциями в комплекте Ред База Данных поставляются и скрипты, содержащие операторы объявления всех функций UDF. Имена скриптов соответствуют именам библиотек dll (so), в которых располагаются сами функции. Имена файлов скриптов имеют расширение sql. Функции и скрипты располагаются в каталоге UDF корневого каталога инсталляции сервера базы данных.

Чтобы сделать функции, определенные пользователем, доступными при работе с вашей базой данных, необходимо выполнить соответствующие скрипты, добавив в самое начало скрипта оператор соединения с вашей базой данных CONNECT.

Пример объявления функции abs в файле скриптов ib_udf.sql:

```
DECLARE EXTERNAL FUNCTION abs
  DOUBLE PRECISION
RETURNS DOUBLE PRECISION BY VALUE
ENTRY_POINT 'IB_UDF_abs' MODULE_NAME 'ib_udf';
```

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.

В настоящей версии Ред База Данных нет необходимости использовать существующие UDF, поскольку большинство из них поставляются в виде встроенных функций.

# Приложение Д 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 Д.1. Синтаксис оператора ALTER CHARACTER SET**

```
ALTER CHARACTER SET <набор символов>
SET DEFAULT COLLATION <сортировка>;
```

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.

```
ALTER CHARACTER SET win1252
SET DEFAULT COLLATION win_ptbr;
```

## ALTER DATABASE

Оператор ALTER DATABASE/SHEMA позволяет добавить к существующей базе данных произвольное количество вторичных файлов. Для выполнения оператора необходимо предварительно соединиться с базой данных (оператор CONNECT). Синтаксис оператора:

**Листинг Д.2. Синтаксис оператора ALTER DATABASE**

```
ALTER {DATABASE | SCHEMA}
  ADD <вторичный файл> [ADD <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]
  [[SET] MAC PLUGIN <модуль мандатного доступа> | DROP MAC PLUGIN];

<вторичный файл> ::=
  FILE '<спецификация файла>'
  [LENGTH [=] <целое> [PAGE[S]]]
  [STARTING [AT [PAGE]] <целое>]

<спецификация файла> ::=
  [{ <имя сервера>: | \\<имя сервера>\ }] { <путь к файлу БД> | <алиас БД> }
```

Изменять базу данных может ее владелец, администратор и пользователь с ролью ALTER DATABASE.

Предложение ADD добавляет к базе данных вторичный файл. Для вторичного файла нужно указать полный путь к файлу и имя вторичного файла.

Предложение LENGTH задает количество страниц во вторичном файле базы данных. Предложение STARTING AT PAGE задает номер страницы, с которой должен начинаться вторичный файл.

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).

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.

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.

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.

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.

Предложение ENCRYPT WITH шифрует базу данных с помощью указанного плагина шифрования (если таковой имеется). Шифрование начинается сразу после этого оператора и будет выполняться в фоновом режиме. Нормальная работа с базами данных не нарушается во время шифрования. Необязательное предложение KEY позволяет передать имя ключа для плагина шифрования. Что делать с этим именем ключа решает плагин.

> Ред База Данных не предоставляет плагин шифрования; его можно написать самому или получить у сторонних разработчиков. Существует возможность только его подключить и воспользоваться им.

Предложение DECRYPT дешифрует базу данных.

Для активации или изменения модуля мандатного доступа у существующей базы указывается предложение [SET] MAC PLUGIN, которое задает необходимый модуль мандатного доступа. После

этого следует перезапустить все подключения. Для отключения контроля доступа в предложении DROP MAC PLUGIN не указывается никакой модуль. Более подробно о мандатном принципе контроля доступа см. в Руководстве администратора.

Подробно оператор описан в главе 3 «Работа с базой данных».

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 Д.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>]
]];
```

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>)), удалить существующее условие домена (предложение DROP CONSTRAINT), задать новый тип данных (TYPE <тип данных>), изменить набор символов (CHARACTER SET <набор символов>) и изменить порядок сортировки (COLLATE <порядок сортировки>).

Подробнее условие домена описано в операторе CREATE DOMAIN в этом приложении.

Изменить существующий домен может владелец домена (его создатель), пользователь с административными привилегиями или пользователь с привилегией ALTER ANY DOMAIN.

Подробно типы данных и оператор ALTER DOMAIN с примерами описаны в главе 4 «Работа с доменами».

См. также операторы CREATE DOMAIN, DROP DOMAIN.

# ALTER EXCEPTION

Оператор позволяет изменить текст существующего в базе данных пользовательского исключения. Синтаксис:

**Листинг Д.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 FUNCTION

Оператор позволяет изменить в функции, определенной пользователем (UDF — User Defined Function), имя точки входа и/или имя модуля. Его синтаксис:

**Листинг Д.5. 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.

См. также оператор DECLARE EXTERNAL FUNCTION, DROP EXTERNAL FUNCTION.

# ALTER FUNCTION

Для изменения существующей хранимой функции используется оператор ALTER FUNCTION. Синтаксис оператора представлен в листинге Д.6.

**Листинг Д.6. 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
   [<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>
                | <объявление 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;
- точку входа и имя движка (для внешних функций)

После выполнения существующие привилегии и зависимости сохраняются.

См. также операторы CREATE OR ALTER FUNCTION, RECREATE FUNCTION, CREATE FUNCTION, DROP FUNCTION, DECLARE VARIABLE, DECLARE CURSOR, DECLARE FUNCTION, DECLARE PROCEDURE.

# ALTER INDEX

Оператора ALTER INDEX позволяет сделать индекс активным или неактивным. Возможностей изменения структуры или упорядоченности его столбцов этот оператор не предусматривает. Его синтаксис:

**Листинг Д.7. Синтаксис оператора ALTER INDEX**

```
ALTER INDEX <имя индекса> {ACTIVE | INACTIVE};
```

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.

Состояние индекса может изменять только владелец таблицы, для которой создан индекс, администратор и пользователь с привилегией ALTER ANY TABLE. Подробнее см. в документе «Руководство администратора».

Подробно оператор описан в главе 7 «Работа с индексами».

См. также операторы CREATE INDEX, DROP INDEX, SET STATISTICS.

# ALTER MAPPING

Оператор ALTER MAPPING позволяет изменять любые опции существующего отображения.

**Листинг Д.8. 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).

См. также операторы CREATE OR ALTER MAPPING, DROP MAPPING, CREATE MAPPING.

# ALTER PACKAGE

Оператор ALTER PACKAGE изменяет заголовок пакета. Синтаксис оператора представлен в листинге Д.9.

**Листинг Д.9. 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>]

<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.

Позволяется изменять количество и состав процедур и функций, их входных и выходных параметров. При этом исходный код тела пакета сохраняется. Состояние соответствия тела пакета его заголовку отображается в таблице RDB$PACKAGES в столбце RDB$VALID_BODY_FLAG. Также данным оператором можно поменять в контексте какого пользователя будет выполняться пакет.

См. также операторы RECREATE PACKAGE, DROP PACKAGE, CREATE PACKAGE, CREATE OR ALTER PACKAGE, CREATE PROCEDURE, CREATE FUNCTION.

# ALTER POLICY

Для изменения политики безопасности администратору необходимо соединиться с какой-либо базой данных. Для изменения политики используется оператор ALTER POLICY. Синтаксис этого оператора приведен ниже:

```
ALTER POLICY <имя политики> AS <параметр>=<значение> [,<параметр>=<значение>...]
```

Возможные параметры политик, а также у каких пользователей есть права на операцию ALTER POLICY можно посмотреть в описании параметра CREATE POLICY.

См. также операторы CREATE POLICY, DROP POLICY.

# ALTER PROCEDURE

Для изменения существующей хранимой процедуры используется оператор ALTER PROCEDURE. Синтаксис оператора:

**Листинг Д.10. Синтаксис оператора ALTER PROCEDURE**

```
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
   [<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>
               | <объявление функции>
```

Оператор позволяет изменять состав и характеристики входных параметров, состав и характеристики выходных параметров, тело хранимой процедуры, а также в контексте какого пользователя будет выполняться процедура.

В одном операторе ALTER PROCEDURE можно изменять любую из перечисленных частей или все сразу.

Выполняемые изменения хранимой процедуры не оказывают никакого влияния на ее зависимости.

Изменять хранимую процедуру может ее создатель и администратор (пользователь с ролью RDB$ADMIN) и пользователь с привилегией ALTER ANY PROCEDURE.

См. также операторы CREATE PROCEDURE, RECREATE PROCEDURE, DROP PROCEDURE, EXECUTE PROCEDURE, DECLARE VARIABLE, DECLARE CURSOR, DECLARE FUNCTION, DECLARE PROCEDURE.

# ALTER ROLE

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.

**Листинг Д.11. Синтаксис оператора ALTER ROLE**

```
ALTER ROLE RDB$ADMIN {SET | DROP} AUTO ADMIN MAPPING
```

Этот оператор может быть выполнен пользователями с достаточными правами, а именно:

- владельцем базы данных;
- администратором.

Нет никаких операторов SQL, чтобы включить или выключить флаг AUTO ADMIN MAPPING в базе данных пользователей. Для этого можно использовать только утилиту командной строки gsec:

```
gsec -mapping set
gsec -mapping drop
```

Включение/выключение флага AUTO ADMIN MAPPING разрешено выполнять только:

- SYSDBA;
- При включенном флаге AUTO ADMIN MAPPING - любой администратор операционной системы Windows (при использовании Trusted Authentication) без указания роли.

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 Д.12. Синтаксис оператора ALTER SEQUENCE**

```
ALTER {SEQUENCE | GENERATOR} <имя генератора>
  [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 6 «Работа с генераторами».

См. также операторы CREATE GENERATOR, CREATE SEQUENCE, DROP GENERATOR, DROP SEQUENCE, SET GENERATOR, функцию GEN_ID(), конструкцию NEXT VALUE FOR.

# ALTER TABLE

Оператор ALTER TABLE используется для изменения описания таблицы, существующей а базе данных. Изменять таблицу может ее владелец, администратор и пользователь с ролью ALTER ANY TABLE.

Синтаксис оператора ALTER TABLE:

**Листинг Д.13. Синтаксис оператора ALTER TABLE**

```
ALTER TABLE <имя таблицы> <операция изменения> [, <операция изменения>...];
```

В одном операторе можно выполнить произвольное количество операций изменений. Синтаксис операции:

```
<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 }
```

Для добавления нового столбца следует ввести в операторе изменения таблицы ALTER TABLE следующую конструкцию:

```
ADD <определение столбца>
```

Синтаксис определения столбца:

```
<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 BY DEFAULT AS IDENTITY [(START WITH <стартовое значение>)]
   [<ограничение столбца>]
```

Для столбца таблицы можно задать тип данных, имя домена, на котором основывается этот столбец, или указать, что этот столбец является вычисляемым (вариант COMPUTED BY или GENERATED ALWAYS AS).

Можно указать значение по умолчанию (DEFAULT), предложение недопустимости пустого значения (NOT NULL), задать ограничения столбца и установить порядок сортировки для символьных столбцов (предложение COLLATE). Синтаксис и семантика того, как в таблице описываются столбцы и их ограничения см. в операторе CREATE TABLE.

Одной из операций изменения таблицы является добавление ограничения таблицы. Синтаксис ограничения таблицы также см. в описании оператора CREATE TABLE.

Для удаления существующего столбца таблицы в операторе изменения таблицы надо ввести:

```
DROP <имя столбца>
```

Прежде чем удалять столбец, нужно удалить все зависимости в базе данных, связанные с этим столбцом.

Чтобы удалить существующее ограничение столбца или таблицы следует в операторе изменения таблицы ввести:

```
DROP CONSTRAINT <имя ограничения столбца или таблицы>
```

При использовании оператора ALTER TABLE есть несколько вариантов изменения характеристик существующего столбца таблицы с помощью предложения ALTER [COLUMN]:

- изменение имени;
- изменение типа данных;
- изменение позиции столбца в списке столбцов таблицы;
- удаление значения по умолчанию столбца;
- добавление значения по умолчанию столбца;
- изменение типа и выражения для вычисляемого столбца;
- изменение столбцов идентификации.

Само предложение ALTER [COLUMN] выглядит следующим образом:

```
ALTER [COLUMN] <имя столбца> <модификация столбца>

<модификация столбца> ::= <модификация столбца> ::= <мод-я обычного столбца> |
<мод-я вычисляемого столбца> | <мод-я идентификационного столбца>

<модификация обычного столбца> ::=
    TO <новое имя столбца>
  | POSITION <новая позиция>
  | TYPE { <тип данных> | <имя домена> }
  | SET DEFAULT { <литерал> | NULL | <контекстная переменная>}
  | DROP DEFAULT
  | SET NOT NULL
  | DROP NOT NULL

<модификация вычисляемого столбца> ::=
    TO <новое имя столбца>
  | POSITION <новая позиция>
  | [TYPE <тип данных>] {GENERATED ALWAYS AS | COMPUTED [BY]} (<выражение>)

<модификация идентификационного столбца> ::=
    TO <новое имя столбца>
  | POSITION <новая позиция>
  | RESTART [ WITH <стартовое значение> ]
```

Подробно оператор описан в главе 5 «Работа с таблицами».
См. также операторы CREATE TABLE, DROP TABLE.

# ALTER TRIGGER

Оператор ALTER TRIGGER позволяет изменять заголовок и/или тело существующего триггера. Синтаксис оператора:

**Листинг Д.14. 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
   [<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 триггером 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 только изменить его активность, позицию и выполняемые действия. 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

Можно управлять учётными записями пользователей средствами операторов SQL. Для изменения существующей учетной записи пользователя используется следующий синтаксис:

**Листинг Д.15. 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.

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.

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.

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.

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.

Предложение GRANT ADMIN ROLE предоставляет указанному пользователю привилегии роли RDB$ADMIN в базе данных безопасности (security3.fdb). 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.

> 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.

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 Д.16. Синтаксис оператора ALTER VIEW**

```
ALTER VIEW <имя представления>
   [( <имя столбца> [AS <псевдоним>] [, <имя столбца> [AS <псевдоним>] ...] )]
```

```
AS <оператор SELECT>
[WITH CHECK OPTION];
```

Изменить представление могут только владелец представления, администратор, пользователь с привилегией ALTER ANY VIEW.

Все предложения в этом операторе в точности соответствуют предложениям в операторе CREATE VIEW.

См. также операторы CREATE OR ALTER VIEW, CREATE VIEW, DROP VIEW, RECREATE VIEW.

# COMMENT

Оператор COMMENT позволяет создавать примечания, комментарии, для объектов базы данных.

**Листинг Д.17. 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> }

<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 Д.18. 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.

Необязательное ключевое слово `WORK` используется 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` применяется для совместимости с предыдущими версиями серверов базы данных.

Если используется предложение `RETAIN [SNAPSHOT]`, то выполняется мягкое подтверждение транзакции. Изменения, выполненные в контексте данной транзакции, становятся доступными другим процессам, работающим с этой базой данных, сама транзакция продолжает оставаться активной.

Если уровень изоляции такой транзакции `SNAPSHOT` или `SNAPSHOT TABLE STABILITY`, то после выполнения 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. Мягкое подтверждение не освобождает ресурсов сервера.

Подробно оператор описан в главе 10 «Транзакции».

См. также операторы `SET TRANSACTION`, `ROLLBACK`, `SAVEPOINT`, `RELEASE SAVEPOINT`.

# CONNECT

Оператор `CONNECT` используется для соединения с существующей базой данных. Синтаксис:

**Листинг Д.19. 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.

В операторе указывается имя первичного файла базы данных. Имя пользователя (предложение `USER`) и его пароль (`PASSWORD`) должны задавать пользователя, описанного в системе. Имя пользователя может содержать до 31 символа. Оно нечувствительно к регистру. Пароль чувствителен к регистру. Может содержать до 32 символов, однако только первые восемь имеют значение.

Предложение `CACHE` задает количество буферов кэш-памяти для соединения. По умолчанию 256.

Предложение `ROLE` задает имя роли, с которой пользователь соединяется с данной базой данных. Имя роли может содержать до 31 символа. Подробности см. в документе «Руководство администратора».

Подробно оператор описан в главе 3 «Работа с базой данных».

См. также операторы `CREATE ROLE`, `DROP ROLE`, `GRANT`, `REVOKE`.

# CREATE COLLATION

Оператор задает новый порядок сортировки для существующего в базе данных набора символов.

**Листинг Д.20. Синтаксис оператора `CREATE COLLATION`**

```
CREATE COLLATION <имя порядка 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> ...]'];
```

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 RedDatabase.

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 «е» 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 Д.1 — Список доступных атрибутов `COLLATION`

| Имя | Значения | 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. |

| Имя | Значения | Валидность | Описание |
|---|---|---|---|
| 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.

Создать новую сортировку может администратор и пользователь с привилегией CREATE COLLATION. Пользователь, создавший сортировку, становится её владельцем.

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 Д.21. Синтаксис оператора CREATE DATABASE**

```
CREATE {DATABASE | SCHEMA} '<спецификация файла>'
  [USER '<имя пользователя>' [PASSWORD '<пароль>']]
  [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>...]]
  [[SET] MAC PLUGIN <модуль мандатного доступа>];

<спецификация файла> ::=
  [{ <имя сервера>: | \\<имя сервера>\ }] { <путь к файлу БД> | <алиас БД> }

<вторичный файл> ::=
  FILE '<спецификация файла>'
  [LENGTH [=] <целое> [PAGE[S]]]
  [STARTING [AT [PAGE]] <целое>]
```

Можно задавать CREATE DATABASE или CREATE SCHEMA. Это синонимы.

Спецификация файла задает полный путь к создаваемому файлу базы данных и имя файла, включая его расширение. Сам файл должен отсутствовать на внешнем носителе.

Предложения USER и PASSWORD задают, соответственно, имя пользователя и его пароль. Это будет владелец базы данных. Имя пользователя может содержать до 31 символа. Оно нечувствительно к регистру. Пароль чувствителен к регистру. Может содержать до 64 символов, однако только первые восемь имеют значение.

Предложение PAGE_SIZE задает размер страницы базы данных. Допустимы значения 4096 (значение по умолчанию), 8192 и 16384.

Предложение LENGTH задает количество страниц в первичном файле базы данных.

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 типов данных в базе данных. Необязательный параметр COLLATION, связанный с набором символов,

позволяет создавать все текстовые столбцы, домены и переменные с указанной последовательностью сортировки, если не указан другой COLLATE.

> COLLATION, используемый по умолчанию для набора символов в базе данных, может быть изменен с помощью ALTER CHARACTER SET.

Ключевое слово DIFFERENCE FILE служит параметром для задания файла. Данный параметр позволяет задать имя дельта-файла, который будет создаваться при выполнении команды ALTER DATABASE BEGIN BACKUPили запуске утилиты nBackup из командной строки.

База данных может располагаться в одном или более файлах. Тогда после описания первичного файла задается описание вторичных файлов. Помимо спецификации вторичного файла можно задать размер вторичного файла в страницах (предложение LENGTH) и номер страницы, с которой должен начинаться вторичный файл (предложение STARTING AT PAGE).

Для активации мандатного доступа при создании базы указывается предложение [SET] MAC PLUGIN, которое задает необходимый модуль мандатного доступа. При подключении к базе указанный модуль загружается сервером и используется для контроля доступа. Более подробно о мандатном принципе контроля доступа см. в Руководстве администратора.

Создать новую базу данных может только администратор и пользователь с привилегией CREATE DATABASE.

Подробно оператор описан в главе 3 «Работа с базой данных».

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 Д.22. 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.

Имя домена должно быть уникальным среди имен доменов создаваемой (или изменяемой) базы данных. Имя не может превышать 31 символа.

Для задания типа данных используется следующий синтаксис:

```
<тип данных> ::= {
    {SMALLINT | INTEGER | BIGINT} [<размерность массива>]
  | BOOLEAN [<размерность массива>]
  | {FLOAT | DOUBLE PRECISION} [<размерность массива>]
  | {DATE | TIME | TIMESTAMP} [<размерность массива>]
  | {DECIMAL | NUMERIC} [(<точность> [ ,<масштаб>])] [<размерность массива>]
  | {CHAR | CHARACTER | CHARACTER VARYING | VARCHAR} [(<размер>)]
      [CHARACTER SET <набор символов>] [<размерность массива>]
  | {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>...]]
```

Тип данных — один из допустимых типов данных. Это единственный обязательный параметр в операторе создания домена.

Для любого типа данных, кроме BLOB, можно указать размерность массива.

При описании символьного домена и домена типа BLOB можно в предложении CHARACTER SET указать набор символов, если требуется набор, отличный от набора символов по умолчанию, установленный для всей базы данных. В предложении COLLATE можно задать порядок сортировки.

Предложение DEFAULT задает значение по умолчанию — что должно быть помещено в столбец таблицы, основанный на этом домене, если пользователь в операторе INSERT, добавляющем данные в таблицу, не укажет значения столбца. Значением по умолчанию может быть литерал, пустое значение NULL. Литералом может быть любая самоопределенная константа соответствующего типа, предварительно определенный литерал или контекстная переменная. Если значение по умолчанию не устанавливается, то подразумевается пустое значение NULL. В значении по умолчанию нельзя задавать выражения.

Предложение NOT NULL указывает, что столбцу, основанному на этом домене, не может присваиваться пустое значение ни в операторе INSERT, ни в операторе UPDATE.

Предложение CHECK задает условие, которому должно удовлетворять значение, помещаемое оператором INSERT в столбец, основанный на этом домене, или изменяемое оператором UPDATE. Предложение неприменимо к доменам типа BLOB. Условие является логическим выражением, которое может возвращать значения TRUE (истина), FALSE (ложь) и UNKNOWN (неопределенное, неизвестное значение).

Условие имеет следующий синтаксис:

```
<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>)
  | EXISTS (<поиск многих>)
  | 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>
}
```

Значением в условии домена может быть литерал, предварительно определенный литерал, контекстная переменная, а также любое правильное выражение. Для построения выражений можно использовать как внутренние функции SQL, так и функции, определенные пользователем (UDF). Выражения (значения) могут присутствовать как в левой, так и в правой части языковых конструкций условий домена.

Синтаксис значения:

```
<значение> ::= {
```

```
    VALUE [[<элемент массива>]]
  | <литерал>
  | <выражение>
  | 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> ...])]
  | NULL
}
```

Ключевое слово `VALUE` является заменителем имени столбца, который будет основан на данном домене.

Литерал — это числовая константа, строковая константа, заключенная в апострофы, литерал даты или времени, предварительно определенный литерал, контекстная переменная. Выражение — любое правильное выражение SQL.

Выражение — любое правильное выражение SQL. Это может быть арифметическое, строковое или логическое выражение. Арифметическое выражение содержит четыре арифметические операции — сложение, вычитание, умножение и деление. Строковое выражение представлено одной строковой операцией конкатенации (`||`). Логическое выражение может содержать операцию отрицания (`NOT`), дизъюнкцию (логическое ИЛИ, ключевое слово `OR`) и конъюнкцию (логическое И, ключевое слово `AND`).

Конструкция `NEXT VALUE FOR <имя генератора>` является аналогом встроенной функции `GEN_ID(<имя генератора>, 1)`. Значение указанного генератора увеличивается на единицу, и конструкция возвращает новое значение.

Значением может быть обращение к функции, определенной пользователем (User Defined Function, UDF — см. Приложение Г).

В качестве значения может быть также указано пустое значение `NULL`.

Обычная встроенная функция — это функция, которой передается один или более параметров, функция не связана с оператором выборки данных `SELECT`. Функция возвращает ровно одно значение. Все встроенные функции описаны в Приложении Е.

Агрегатные функции в операторе `SELECT` — это функции, определенные в языке SQL Ред База Данных. Агрегатные функции работают с группой значений, полученных при выполнении оператора `SELECT` из таблицы или представления базы данных. Агрегатные функции используются внутри списка выбора оператора `SELECT`. Синтаксис:

```
<агрегатная функция в операторе SELECT> ::=
   SELECT {
       COUNT ({[ALL | DISTINCT] <выражение> | *})
     | SUM ([ALL | DISTINCT] <выражение>)
     | AVG ([ALL | DISTINCT] <выражение>)
     | MAX ([ALL | DISTINCT] <выражение>)
     | MIN ([ALL | DISTINCT] <выражение>)
     | LIST ([ALL | DISTINCT] <выражение>) [, '<разделитель>']
     | CORR (<выражение1>, <выражение2>)
     | COVAR_POP (<выражение1>, <выражение2>)
     | COVAR_SAMP (<выражение1>, <выражение2>)
     | STDDEV_POP (<выражение>)
     | STDDEV_SAMP (<выражение>)
     | VAR_POP (<выражение>)
     | VAR_SAMP (<выражение>)
     | REGR_AVGX (y, x)
```

```
    | REGR_AVGY (y, x)
    | REGR_COUNT (y, x)
    | REGR_INTERCEPT (y, x)
    | REGR_R2 (y, x)
    | REGR_SLOPE (y, x)
    | REGR_SXX (y, x)
    | REGR_SXY (y, x)
    | REGR_SYY(y, x) }
  <предложение FROM>
  [<предложение WHERE>]
```

Оператор SELECT выбирает из указанной таблицы (предложение FROM) на основании некоторого условия, если указано (предложение WHERE), некоторое количество заданных значений. Агрегатная функция внутри оператора SELECT выполняет соответствующие действия и возвращает одно число.

Синтаксис и краткое описание каждой из агрегатных функций описаны в Приложении E.

Оператор VALUE [NOT] IN (<значение> [, <значение>]...) в условии домена указывает, что значение, вводимое в столбец, основанный на этом домене, должно находиться (или не находиться, если указано ключевое слово NOT) в заданном списке.

Оператор VALUE [NOT] BETWEEN <значение> AND <значение> требует, чтобы значение находилось (или не находилось, если указано ключевое слово NOT) в указанном диапазоне, включая граничные значения.

Оператор VALUE [NOT] LIKE <значение> [ESCAPE '<символ>'] задает проверку наличия (или отсутствия в случае указания ключевого слова NOT) во вводимом значении символьного типа данных определенных символов. Ключевое слово ESCAPE позволяет задать символ, который даст возможность использовать шаблонные символы (знак процента % и подчеркивание _) как обычные символы строки, а не как шаблонные символы.

Оператор SIMILAR TO проверяет соответствие вводимого значения с шаблоном регулярного выражения SQL.

В операторе VALUE IS [NOT] NULL проводится проверка вводимых данных на пустое значение. Возвращаемыми значениями могут быть только TRUE и FALSE.

Оператор IS [NOT] DISTINCT FROM выполняется проверка на равенство (неравенство) указанному значению. Отличие от обычных операторов сравнения в том, что два пустых значения NULL считаются равными. Возвращаемыми значениями здесь также могут быть только TRUE и FALSE.

При использовании функций ALL, SOME, ANY используется оператор сравнения. Аргументом любой из функций является оператор SELECT, возвращающий произвольное количество значений одного столбца. Допустимо также и пустое значение.

Функция ALL вернет значение «истина», если сравнение будет истинным для всех значений столбца, полученных из оператора SELECT.

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

Аргументом функции EXISTS является оператор SELECT, возвращающий произвольное количество любых столбцов таблицы. Результатом будет «истина», если оператор SELECT вернет хотя бы одно значение, соответствующее условиям поиска, заданным в предложении WHERE.

Аргументом функции SINGULAR является оператор SELECT, возвращающий произвольное количество любых столбцов таблицы. Результатом будет «истина», если оператор SELECT вернет в точности одно значение, соответствующее условиям поиска, заданным в предложении WHERE.

Вариант VALUE [NOT] CONTAINING <значение> задает проверку на присутствие во вводимом значении (или отсутствие в случае NOT) указанных символов. Символы могут располагаться в любом месте вводимой строки. Этот оператор нечувствителен к регистру. В операторе нельзя использовать шаблонные символы % и _.

Вариант VALUE [NOT] STARTING [WITH] <значение> задает проверку на то, что вводимая строка начинается (или не начинается) с указанных символов. Оператор чувствителен к регистру.

Подробно типы данных и оператор CREATE DOMAIN описаны в главе 4 «Работа с доменами».

См. также операторы ALTER DOMAIN, DROP DOMAIN.

# CREATE EXCEPTION

Оператор создает пользовательское исключение. Синтаксис оператора:

**Листинг Д.23. Синтаксис оператора CREATE EXCEPTION**

```
CREATE EXCEPTION <имя исключения> '<текст сообщения>';
```

Имя исключения может содержать до 31 символа 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_INVALID_VALUE 'Неверное значение @1 для поля @2' ;
```

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, синтаксис которого представлен в листинге Д.24.

**Листинг Д.24. 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
   [<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>
                 | <объявление функции>
```

Хранимую функцию может создать администратор и пользователь с привилегией CREATE FUNCTION. Пользователь, создавший хранимую функцию, становится её владельцем.

Имя хранимой функции должно быть уникальным среди имён всех хранимых функций и внешних (UDF) функций. Для внутренних функций достаточно уникальности только в рамках модулей, которые их «охватывают».

Хранимой функции от вызвавшей программы могут передаваться входные параметры. 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 строкового типа существует возможность задать порядок сортировки с помощью предложения COLLATE. Кроме того, для параметра можно указать ограничение NOT NULL, тем самым запретив передавать в него значение NULL.

Входные 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.

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).

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).

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.

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.

В теле хранимой функции может быть описано произвольное количество локальных переменных, именованных курсоров и подпрограммы (подпроцедуры и подфункции).

После описания локальных переменных в теле хранимой функции следует блок операторов, заключенных в операторные скобки BEGIN и END.

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 Д.25. Синтаксис оператора CREATE GENERATOR**

```
CREATE {GENERATOR | SEQUENCE} <имя генератора>
  [START WITH <начальное значение>] [INCREMENT [BY] <приращение>];
```

Ключевые слова GENERATOR и SEQUENCE являются синонимами.

Имя генератора должно быть уникальным среди имен всех генераторов базы данных и должно содержать до 31 символа.

В момент создания последовательности ей устанавливается значение, указанное в необязательном предложении START WITH. Если предложение START WITH отсутствует, то последовательности устанавливается значение равное 0.

Необязательное предложение INCREMENT [BY] позволяет задать шаг приращения для оператора NEXT VALUE 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 предложении INCREMENT BY.

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 6 «Работа с генераторами».

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 Д.26. Синтаксис оператора создания глобальной временной таблицы**

```
CREATE GLOBAL TEMPORARY TABLE <имя таблицы>
(<определение столбца> [, <определение столбца> | <ограничение таблицы>] ...)
[ON COMMIT {DELETE | PRESERVE} ROWS];
```

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 5 «Работа с таблицами».

См. также операторы CREATE TABLE, ALTER TABLE, DROP TABLE.

# CREATE INDEX

Оператор позволяет создать индекс для таблицы базы данных. Его синтаксис:

**Листинг Д.27. Синтаксис оператора CREATE INDEX**

```
CREATE [UNIQUE] [ASC[ENDING] | DESC[ENDING]]
INDEX <имя индекса> ON <таблица>
{(<столбец> [, <столбец> ...]) | COMPUTED BY (<выражение>)};
```

Индекс создается для одной конкретной таблицы. Имя индекса должно быть уникальным среди имен всех индексов базы данных.

Создать индекс может только владелец таблицы, для которой создан индекс, администратор и пользователь с привилегией ALTER ANY TABLE.

Индекс может быть уникальным (ключевое слово UNIQUE). Это означает, что в индексе не может быть двух разных строк с одинаковыми значениями столбцов. Столбцы, входящие в состав уникального индекса, не могут также иметь пустого значения NULL.

Ключевое слово ASCENDING (сокращенно ASC) означает, что записи индекса упорядочиваются по возрастанию значений столбцов, входящих в состав индекса. Этот вариант по умолчанию.

Ключевое слово DESCENDING (сокращение DESC) указывает, что записи индекса упорядочиваются по уменьшению значений столбцов индекса.

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.

В состав индекса не могут входить вычисляемые поля, а также столбцы, имеющие тип данных BLOB и столбцы любого типа данных, являющиеся массивами.

Подробно оператор описан в главе 7 «Работа с индексами».

См. также операторы ALTER INDEX,DROP INDEX, SET STATISTICS.

# 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 Д.28. 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>]
```

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 являются разными объектами.

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).

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 Д.29. Синтаксис оператора CREATE OR ALTER EXCEPTION**

```
CREATE OR ALTER EXCEPTION <имя исключения> '<текст сообщения>';
```

Имя исключения может содержать до 31 символа и должно быть уникальным среди всех имен исключений базы данных.

Текст сообщения — текст, выдаваемый в момент вызова исключения. Может содержать до 1021 символа.

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 представлен в листинге Д.30.

**Листинг Д.30. 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
   [<объявление> [<объявление> ...] ]
  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 Д.31. 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>]
```

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 представлен в листинге Д.32.

**Листинг Д.32. 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> ::=
   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 Д.33. 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
   [<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>
                 | <объявление 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 Д.34. 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.

# CREATE OR ALTER TRIGGER

Оператор CREATE OR ALTER TRIGGER создает новый триггер для таблицы или представления, если триггера с таким именем не существует в базе данных. Иначе существующий триггер заменяется на новый синтаксис оператора:

**Листинг Д.35. 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>
 | <объявление DDL триггера> }
[SQL SECURITY {DEFINER | INVOKER}]
{ EXTERNAL NAME '<внешний модуль>' ENGINE <имя движка> } |
{
  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
 | 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>
                | <объявление 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 Д.36. 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 Д.37. Синтаксис оператора CREATE OR ALTER VIEW**

```
CREATE OR ALTER VIEW <имя представления>
   [( <имя столбца> [AS <псевдоним>] [, <имя столбца> [AS <псевдоним>] ...] )]
AS <оператор SELECT>
[WITH CHECK OPTION];
```

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 представлен в листинге Д.38.

**Листинг Д.38. 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. Имена процедур и функций, объявленные в заголовке пакета, должны быть уникальны среди имён процедур и функций, объявленных в заголовке и теле пакета.

Подробное описание заголовков хранимых процедур см. в операторе CREATE PROCEDURE, хранимых функций – в операторе CREATE FUNCTION.

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.

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 представлен в листинге Д.39.

**Листинг Д.39. 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
         [<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
         [<объявление> [<объявление> ...] ]
       BEGIN
         <блок операторов>
       END }

<входной параметр> ::= <описание параметра> [{=|DEFAULT} <значение по умолчанию>]
<входной_параметр> ::= <описание параметра>
<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>
                 | <объявление 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.

См. также операторы CREATE PACKAGE, RECREATE PACKAGE BODY, DROP PACKAGE BODY, CREATE PROCEDURE, CREATE FUNCTION.

# CREATE POLICY

Для создания политики безопасности администратору необходимо соединиться с какой-либо базой данных. 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 security3.fdb, создать их можно, соединившись с любой базой данных. Однако пользователь при этом должен иметь права на запись в базу данных безопасности security3.fdb.

Права на запись в базу данных пользователей предоставлены 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);
- При включенном флаге AUTO ADMIN MAPPING в базе данных пользователей (security3.fdb или той, что установлена для вашей базы данных в файле databases.conf) — 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.

Возможные параметры политик следующие:

- AUTH_FACTORS — факторы аутентификации (представлены в таблице Д.2);
- 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_UNUSED_DAYS — максимальное время неактивности учетных записей пользователя, в днях.

Таблица Д.2 — Символическое обозначение факторов аутентификации

| Символическое обозначение | Расшифровка |
|---|---|
| CERT_X509 | Сертификат пользователя |
| PASSWORD | Пароль |

Следующий пример демонстрирует создание политики:

```
CREATE POLICY TestPolicy AS
    AUTH_FACTORS = (CERT_X509, PASSWORD),
    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_UNUSED_DAYS = 45;
```

См. также операторы ALTER POLICY, DROP POLICY.

# CREATE PROCEDURE

Для создания хранимой процедуры используется оператор CREATE PROCEDURE. Синтаксис оператора:

**Листинг Д.40. 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
   [<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>
                 | <объявление функции>
```

Хранимую процедуру может создать администратор и пользователь с привилегией `CREATE PROCEDURE`.

Имя хранимой процедуры может содержать до 31 символа и должно быть уникальным среди имен хранимых процедур базы данных, таблиц и представлений.

Хранимой процедуре от вызвавшей программы могут передаваться входные параметры. 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.

Хранимая процедура может возвращать вызвавшей программе произвольное количество выходных 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.

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.

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.

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 в следующих версиях Ред Базы Данных.

Используйте следующие предложения, чтобы процедура выполнялась:

- с правами вызывающего ее пользователя (значение по умолчанию)

```
SQL SECURITY INVOKER | AUTHID CALLER
```

- с правами ее владельца (создателя)

```
SQL SECURITY DEFINER | AUTHID OWNER
```

Значение по умолчанию на уровне всей базы данных можно изменить оператором `ALTER DATABASE SET DEFAULT SQL SECURITY`.

В теле хранимой процедуры может быть описано произвольное количество локальных пере-

менных, именованных курсоров и подпрограммы (подпроцедуры и подфункции).

После описания локальных переменных в теле хранимой процедуры следует блок операторов, заключенных в операторные скобки BEGIN и END.

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 Д.41. Синтаксис оператора CREATE ROLE**

```
CREATE ROLE <имя роли>;
```

Имя роли может содержать до 31 символа и должно быть уникальным среди всех имен ролей.

Роли 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.

Создать новую роль может администратор и пользователь с привилегией CREATE ROLE.

См. также операторы DROP ROLE, GRANT, REVOKE, CONNECT.

# CREATE SEQUENCE

Другое название для оператора создания генератора. См. в этом приложении CREATE GENERATOR.

**Листинг Д.42. 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 6 «Работа с генераторами».

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 Д.43. 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.

Номер оперативной копии — положительное число, идентифицирующее набор файлов данной оперативной копии.

Если задан вариант AUTO (значение по умолчанию), то в случае, когда оперативная копия становится недоступной, прекращается использование этой оперативной копии, все ссылки на нее удаляются из базы данных. Работа с базой данных продолжается обычным образом без выполнения оперативного копирования.

В случае задания варианта 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 оператором DROP SHADOW. Администратор базы данных должен удалить ссылку на оперативную копию из базы данных, удалить все файлы этой оперативной копии с диска и создать новую оперативную копию.

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 роль основной базы данных.

Предложение LENGTH задает размер оперативной копии в страницах базы данных. Размер страницы оперативной копии равен размеру страницы основного файла базы данных.

Оперативная копия может состоять из нескольких файлов. Для вторичного файла можно задать предложение LENGTH, которое указывает размер файла в страницах базы данных. Предложение STARTING AT PAGE задает номер страницы, с которой должен начинаться вторичный файл после заполнения предыдущих файлов оперативной копии.

Подробно оператор описан в главе 3 «Работа с базой данных».

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 Д.44. Синтаксис оператора CREATE TABLE**

```
CREATE TABLE <имя таблицы>
  [EXTERNAL [FILE] '<спецификация файла>' [ADAPTER 'CSV']]
  (<определение столбца> [, {<определение столбца> | <ограничение таблицы>}...])
  [SQL SECURITY {DEFINER | INVOKER}];
```

Имя таблицы должно быть уникальным среди имен таблиц базы данных, хранимых процедур и представлений, описанных в этой базе данных.

Таблица может содержать, по меньшей мере, один столбец и произвольное количество ограничений таблицы.

Таблица может не храниться в базе данных, все ее строки могут помещаться в отдельный текстовый файл, находящийся вне базы данных, заданный предложением EXTERNAL FILE в операторе создания таблицы. СУБД Ред База Данных поддерживает два формата внешних файлов: формат «строк» с фиксированной длиной и .csv формат.

В таблицах, которые хранятся во внешних файлах, могут быть описаны любые типы данных, кроме BLOB. Недопустимо также использование массивов с любым типом данных. По отношению к таблицам, хранящимся во внешних файлах, допустимы только операции добавления новых строк (INSERT) и выборки (SELECT) данных. Для .csv формата допустима только выборка данных. Опера-

ции же изменения существующих данных (UPDATE) или удаления строк такой таблицы (DELETE) не могут быть выполнены. Возможность использования для таблиц внешних файлов зависит от установки значения параметра ExternalFileAccess в файле конфигурации firebird.conf. Подробности см. в главе 5 «Работа с таблицами».

Необязательное предложение SQL SECURITY {DEFINER | INVOKER} определяет, в контексте какого пользователя будет проходить работа с таблицей. Ключевое слово INVOKER (значение по умолчанию) указывает, что таблица вызывается с правами текущего пользователя. Задание ключевого слова DEFINER означает, что таблица вызывается 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.

Создать новую таблицу может администратор и пользователь с привилегией CREATE TABLE. Пользователь, создавший таблицу, становится её владельцем.

## Определение столбца

Таблица должна содержать не менее одного столбца. Синтаксис описания столбца таблицы:

**Листинг Д.45. Синтаксис описания столбца таблицы**

```
<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 BY DEFAULT AS IDENTITY [(START WITH <стартовое значение>)]
   [<ограничение столбца>]
```

Столбец должен иметь имя, уникальное в данной таблице.

Для столбца обязательно должен быть указан либо тип данных, либо имя домена, характеристики которого будут скопированы в этот столбец, либо должно быть указано, что столбец является вычисляемым (COMPUTED BY или GENERATED ALWAYS AS).

Синтаксис задания типа данных:

```
<тип данных> ::= {
    {SMALLINT | INTEGER | BIGINT} [<размерность массива>]
  | BOOLEAN [<размерность массива>]
  | {FLOAT | DOUBLE PRECISION} [<размерность массива>]
  | {DATE | TIME | TIMESTAMP} [<размерность массива>]
  | {DECIMAL | NUMERIC} [(<точность> [ ,<масштаб>])] [<размерность массива>]
  | {CHAR | CHARACTER | CHARACTER VARYING | VARCHAR} [(<размер>)]
      [CHARACTER SET <набор символов>] [<размерность массива>]
  | {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.

При описании символьного столбца и столбца с типом данных BLOB можно в предложении CHARACTER SET указать набор символов, если требуется набор, отличный от набора символов по умолчанию, установленного для базы данных. В предложении COLLATE можно задать порядок сортировки (для типа данных BLOB использование COLLATE недопустимо).

Подробное описание типов данных приведено в главе 2 «Типы данных Ред База Данных».

Если вместо типа данных задается имя домена, то все его характеристики копируются для этого столбца.

Вычисляемый столбец задается предложением COMPUTED BY. Значение такого столбца не хранится в таблице, а вычисляется каждый раз при выборке данных.

Столбцы идентификации могут быть определены с помощью предложения GENERATED BY DEFAULT AS IDENTITY. Столбец идентификации представляет собой столбец, связанный с внутренним генератором последовательностей. Его значение устанавливается автоматически каждый раз, когда оно не указано в операторе INSERT. Необязательное предложение START WITH позволяет указать начальное значение отличное от нуля. 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).

Необязательное предложение DEFAULT определяет значение по умолчанию для столбца — то значение, которое будет присвоено столбцу, если при помещении новой строки в таблицу в операторе INSERT не указан данный столбец и его значение. Значением по умолчанию может быть литерал, пустое значение NULL. Литералом может быть любая самоопределенная константа соответствующего типа, предварительно определенный литерал или контекстная переменная. Если значение по умолчанию явно не устанавливается, то подразумевается пустое значение, NULL. Использование выражений в значении по умолчанию недопустимо.

Необязательное предложение NOT NULL указывает, что столбцу не может быть присвоено пустое значение.

Ограничение столбца записывается после определения других характеристик столбца. Существует четыре вида ограничений столбца: первичный ключ (PRIMARY KEY), уникальный ключ (UNIQUE), внешний ключ (REFERENCES) и ограничение условия столбца CHECK. Синтаксис ограничения столбца:

```
<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>) }

<предложение USING> ::= USING [ASC[ENDING] | DESC[ENDING]] INDEX <имя индекса>
```

Необязательное предложение CONSTRAINT задает имя ограничения. Если задано это предложение, то система автоматически строит индекс с этим именем для поддержания соответствующего ограничения (если в ограничении не было задано предложение USING). Иначе ограничению и индексу присваивается системное имя.

Ограничение UNIQUE определяет уникальный ключ: значение столбца должно быть уникальным — в таблице не должно существовать двух разных строк, имеющих одно и то же значение этого столбца. Уникальный ключ в отличие от первичного может иметь пустое значение NULL. Строк с

пустым значением уникального ключа в таблице может быть произвольное количество. Таблица может содержать любое количество уникальных ключей. Для уникального ключа система автоматически создает индекс. Если ограничению при этом было назначено имя в предложении CONSTRAINT (при отсутствии предложения USING), то это имя присваивается созданному индексу. Если же было задано и предложение USING, то индекс для уникального ключа получает имя, указанное в этом предложении. В противном случае система присваивает индексу системное имя.

Ограничение PRIMARY KEY определяет первичный ключ. В таблице может быть только один первичный ключ. Столбец, являющийся первичным ключом, обязательно должен быть описан с указанием NOT NULL. Для первичного ключа система автоматически создает соответствующий индекс. Если ограничению было назначено имя в предложении CONSTRAINT (при отсутствии предложения USING), то это имя присваивается созданному индексу. Если же было задано и предложение USING, то индекс для первичного ключа получает имя, указанное в этом предложении.

Ограничение REFERENCES определяет внешний ключ. Внешний ключ должен иметь пустое значение NULL или же он должен ссылаться на первичный или уникальный ключ другой или той же самой таблицы (родительской таблицы). Для внешнего ключа система автоматически создает индекс. Если ограничению было назначено имя в предложении CONSTRAINT (при отсутствии предложения USING), то это имя присваивается созданному индексу. Если же было задано и предложение USING, то индекс для внешнего ключа получает имя, указанное в этом предложении.

Предложение USING позволяет задать имя индекса для поддержания соответствующего ограничения и указать его упорядоченность — по возрастанию значений реквизитов (ASCENDING) или по убыванию их значений (DESCENDING). Если упорядоченность не задана, то предполагается ASCENDING, по возрастанию. Предложение USING может быть использовано для ограничений первичного ключа (PRIMARY KEY), уникального ключа (UNIQUE) и внешнего ключа (REFERENCES). Упорядоченность индекса внешнего ключа должна соответствовать упорядоченности индекса первичного (уникального) ключа, на который ссылается данный внешний ключ. Для ограничения CHECK предложение USING не применимо.

Предложение ON DELETE определяет, что произойдет с записями подчиненной таблицы при удалении соответствующей строки главной таблицы:

- NO ACTION — не будет выполнено никаких действий; в клиентской программе должны быть предприняты специальные меры по поддержанию ссылочной целостности данных. Это может выполнять сама клиентская программ или для этого следует написать выполняемую хранимую процедуру, к которой должно осуществляться в процессе удаления строки обращение из клиентской программы, или специально созданный пользовательский триггер до удаления (BEFORE DELETE), выполняющий все необходимые установки;
- CASCADE — в подчиненной таблице должны быть удалены все записи, имеющие те же значения внешнего ключа, что и значение первичного (уникального) ключа удаленной строки главной таблицы;
- SET DEFAULT — значения внешнего ключа всех соответствующих строк в подчиненной таблице устанавливаются в значение по умолчанию, определенное в предложении DEFAULT этого столбца, описанного как внешний ключ. Если значение по умолчанию не указано, то столбцу присваивается значение NULL;
- SET NULL — значения внешнего ключа всех соответствующих строк в подчиненной таблице устанавливаются в пустое значение NULL.

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 подчиненной таблицы при изменении значения первичного/уникального ключа в строке главной таблицы:

- NO ACTION — не будет выполнено никаких действий; в программе должны быть предприняты специальные меры по поддержанию ссылочной целостности данных;
- CASCADE — в подчиненной таблице должны быть изменены все значения внешнего ключа, имеющие те же значения, что и значение первичного (уникального) ключа изменяемой

строки главной таблицы;

- SET DEFAULT — значения внешнего ключа всех соответствующих строк в подчиненной таблице устанавливаются в значение по умолчанию, заданное в предложении DEFAULT для этого столбца;
- SET NULL — значения внешнего ключа всех соответствующих строк в подчиненной таблице устанавливаются в пустое значение NULL.

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.

Ограничение CHECK определяет условие, которому должно удовлетворять значение, помещаемое в данный столбец. Это логическое выражение, которое может возвращать значения TRUE (истина), FALSE (ложь) и UNKNOWN (неопределенное, неизвестное значение). Это условие используется при добавлении в таблицу новой строки (оператор INSERT) и при изменении существующего значения столбца таблицы (операторы UPDATE или UPDATE OR INSERT). Синтаксис условия столбца:

```
<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>)
  | EXISTS (<поиск многих>)
  | 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> }
```

Оператором в этом условии является оператор сравнения.

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

В операторе сравнения символы «!» и «^» означают отрицание. Оператор может быть применен к любому типу данных, за исключением типа данных BLOB. Допустимо сравнение однотипных или близких типов данных. При необходимости следует выполнить явное преобразование типа данных операндов сравнения, используя функцию CAST.

| **Операторы** | = | <> , != , ^= | > | < | >= , !< , ^< | <= , !> , ^> |
|---|---|---|---|---|---|---|
| **Значение** | Равно | Не равно | Больше | Меньше | Больше или равно, не меньше | Меньше или равно, не больше |

```
<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 }
```

В качестве значения можно указать имя столбца таблицы. Если столбец является массивом, то в квадратных скобках нужно указать конкретный элемент массива.

Литерал — числовая константа, строковая константа, заключенная в апострофы, литерал даты или времени, предварительно определенный литерал, контекстная переменная (см. главу 2 «Типы данных Ред База Данных»).

Обычная встроенная функция — это функция, получающая один или более входных параметров, которая не связана с оператором SQL выборки данных `SELECT`. Функция возвращает ровно одно значение. Встроенные функции описаны в Приложении Е.

Агрегатные функции в операторе `SELECT` — функции, определенные в языке SQL. Это агрегатные функции, работающие не с одним фиксированным набором параметров, а с группой значений, полученных при выполнении оператора `SELECT` из таблицы базы данных. Агрегатные функции используются внутри списка выбора оператора `SELECT`. Синтаксис:

```
<агрегатная функция в операторе SELECT> ::=
   SELECT {
       COUNT ({[ALL | DISTINCT] <выражение> | *})
     | SUM ([ALL | DISTINCT] <выражение>)
     | AVG ([ALL | DISTINCT] <выражение>)
     | MAX ([ALL | DISTINCT] <выражение>)
     | MIN ([ALL | DISTINCT] <выражение>)
     | LIST ([ALL | DISTINCT] <выражение>) [, '<разделитель>']
     | CORR (<выражение1>, <выражение2>)
     | COVAR_POP (<выражение1>, <выражение2>)
     | COVAR_SAMP (<выражение1>, <выражение2>)
     | STDDEV_POP (<выражение>)
     | STDDEV_SAMP (<выражение>)
     | VAR_POP (<выражение>)
     | VAR_SAMP (<выражение>)
     | REGR_AVGX (y, x)
     | REGR_AVGY (y, x)
     | REGR_COUNT (y, x)
     | REGR_INTERCEPT (y, x)
     | REGR_R2 (y, x)
     | REGR_SLOPE (y, x)
     | REGR_SXX (y, x)
     | REGR_SXY (y, x)
     | REGR_SYY(y, x) }
   <предложение FROM>
   [<предложение WHERE>]
```

Оператор `SELECT` выбирает из указанной таблицы (предложение `FROM`) на основании некоторого условия, если указано (предложение `WHERE`), некоторое количество заданных значений. Агрегатная функция внутри оператора `SELECT` выполняет соответствующие действия и возвращает одно число.

Синтаксис и краткое описание каждой из агрегатных функций описаны в Приложении Е.

Оператор IN в условии столбца указывает, что вводимое в столбец значение должно находиться (или не находиться, если указано ключевое слово NOT) в заданном списке.

В операторе BETWEEN проверяется, присутствует ли значение, записанное в левой части условия, в диапазоне, заданном в правой части условия, включая граничные значения.

Оператор LIKE задает проверку наличия (или отсутствия в случае указания ключевого слова NOT) во вводимом значении символьного типа данных определенных символов.

Оператор IS [NOT] NULL осуществляет проверку на пустое значение (или отсутствие пустого значения). Он возвращает только значения TRUE или FALSE.

Оператор IS [NOT] DISTINCT FROM проверки на равенство (неравенство, если задано NOT). В отличие от операторов равно и не равно этот оператор трактует два пустых значения NULL как равные друг другу. Как и в случае оператора IS [NOT] NULL данный оператор всегда возвращает либо TRUE, либо FALSE.

При использовании функций ALL, SOME, ANY используется оператор сравнения. Аргументом каждой из функций является оператор SELECT, возвращающий произвольное количество значений одного столбца. Допустимо также и пустое значение.

Функция ALL вернет значение «истина», если сравнение будет истинным для всех значений столбца, полученных из оператора SELECT.

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

Аргументом функции EXISTS является оператор SELECT, возвращающий произвольное количество любых столбцов таблицы. Результатом будет «истина», если оператор SELECT вернет хотя бы одно значение, соответствующее условиям поиска, заданным в предложении WHERE.

Аргументом функции SINGULAR является оператор SELECT, возвращающий произвольное количество любых столбцов таблицы. Результатом будет «истина», если оператор SELECT вернет в точности одно значение, соответствующее условиям поиска, заданным в предложении WHERE.

Результатом оператора CONTAINING будет «истина», если значение в левой части выражения будет содержать в качестве своей части значение, указанное в правой части. Этот оператор не чувствителен к регистру.

Результатом оператора STARTING WITH будет «истина», если значение в левой части выражения будет начинаться с символов, указанных в правой части. Оператор чувствителен к регистру.

## Ограничение таблицы

Ограничения таблицы являются более универсальным способом описания ограничений, применяемым только к одному столбцу или к группе столбцов таблицы.

Ограничение таблицы описывается следующим синтаксисом:

```
<ограничение таблицы> ::=
  [CONSTRAINT <имя ограничения>]
  {
    PRIMARY KEY (<имя столбца> [, <имя столбца> ...]) [<предложение USING>]
  | UNIQUE (<имя столбца> [, <имя столбца> ...]) [<предложение USING>]
  | FOREIGN KEY (<имя столбца> [, <имя столбца> ...])
    REFERENCES <имя таблицы> (<имя столбца> [, <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>)
  }

<предложение USING> ::= USING [ASC[ENDING] | DESC[ENDING]] INDEX <имя индекса>
```

Ограничение таблицы, в отличие от ограничения столбца, может относиться не только к отдельному столбцу, но и к группе столбцов таблицы.

Предложение PRIMARY KEY задает ограничение первичного ключа. В состав первичного ключа в ограничении таблицы может входить один или более столбцов данной таблицы.

Предложение UNIQUE задает ограничение уникального ключа.

Предложение FOREIGN KEY задает ограничение внешнего ключа. Синтаксис этого ограничения на уровне таблицы несколько отличается от синтаксиса ограничения на уровне столбца:

```
FOREIGN KEY (<имя столбца> [, <имя столбца> ...])
REFERENCES <имя таблицы> (<имя столбца> [,<имя столбца>...])[<предложение USING>]
   [ON DELETE { NO ACTION | CASCADE | SET DEFAULT | SET NULL }]
   [ON UPDATE { NO ACTION | CASCADE | SET DEFAULT | SET NULL }]
```

После ключевых слов FOREIGN KEY указывается список столбцов создаваемой таблицы, которые входят в состав внешнего ключа. Список заключается в круглые скобки.

После ключевого слова REFERENCES указывается имя родительской таблицы, на первичный или уникальный ключ которой ссылается описываемый внешний ключ. Список имен столбцов родительской таблицы, входящих в состав первичного (уникального) ключа помещается сразу после имени таблицы и заключается в круглые скобки. Структура внешнего ключа дочерней таблицы по количеству столбцов и по типам данных, включая размерность символьных данных, должна полностью соответствовать структуре первичного (уникального) ключа родительской таблицы. Совпадения имен не требуется. Упорядоченность индекса внешнего ключа должна соответствовать упорядоченности индекса первичного (уникального) ключа, на который ссылается данный внешний ключ.

Необязательные предложения ON DELETE и ON UPDATE определяют, что будет происходить с дочерней таблицей, соответственно, при удалении или изменении данных в родительской таблице. Описание соответствующих действий см. ранее в этом операторе, где рассматривались ограничения столбца.

Ограничение CHECK задает условия, которым должны удовлетворять значения столбцов данной таблицы при помещении в таблицу новой строки или при изменении значений отдельных столбцов существующей строки таблицы.

Варианты и правила использования условий таблицы полностью совпадают с условиями столбца, описанными ранее в этом операторе.

Подробно оператор описан в главе 5 «Работа с таблицами».

См. также операторы ALTER TABLE, DROP TABLE.

# CREATE TRIGGER

Оператор CREATE TRIGGER создает новый триггер для таблицы или представления. Синтаксис:

**Листинг Д.46. 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
   [<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
 | 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>
                | <объявление функции>
```

Табличный триггер может быть создан владельцем таблицы (представления), для которого создает-

ся 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.

Имя триггера может содержать до 31 символа и должно быть уникальным среди имен всех триггеров базы данных.

Триггер может быть создан как для таблицы (представления) базы данных, так и для события базы данных.

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.

Если триггер создается для таблицы (представления), то для него указывается событие и фаза события.

Ключевое слово BEFORE означает, что триггер вызывается до наступления соответствующего события (событий, если их указано несколько), AFTER — после наступления события (событий).

Для триггера может быть указано одно из событий таблицы (представления) — INSERT (добавление), UPDATE (изменение), DELETE (удаление) — или несколько событий, при которых вызывается триггер. Разделителем списка событий является ключевое слово OR.

Для одного и того же события или группы событий одной таблицы (представления) может быть создано несколько триггеров. Ключевое слово POSITION позволяет задать порядок, в котором будут выполняться такие триггеры (по умолчанию значение 0). Если позиции для триггеров не заданы или несколько триггеров имеют одно и то же значение позиции, то такие триггеры будут выполняться в алфавитном порядке их имен.

Триггер может вызываться при наступлении событий базы данных — при соединении с базой данных (CONNECT), при отсоединении от базы данных (DISCONNECT), при старте транзакции (TRANSACTION START), при подтверждении транзакции (TRANSACTION COMMIT) и при откате транзакции (TRANSACTION ROLLBACK). Порядок вызова таких триггеров также соответствует правилам, установленным для триггеров, вызываемых при наступлении события таблицы (представления). Нельзя создавать один триггер для нескольких событий базы данных.

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.

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.

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.

В теле триггера может быть описано произвольное количество локальных переменных, именованных курсоров и подпрограммы (подпроцедуры и подфункции).

После описания локальных переменных в теле триггера следует блок операторов, заключенных в операторные скобки BEGIN и END.

См. также операторы ALTER TRIGGER, CREATE OR ALTER TRIGGER, RECREATE TRIGGER, DROP TRIGGER.

# CREATE USER

Можно управлять учётными записями пользователей средствами операторов SQL. Для создания новой учетной записи пользователя используется следующий синтаксис:

**Листинг Д.47. 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 (`security3.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.

Для создания учетной записи пользователя текущий пользователь должен обладать административными привилегиями в базе даных безопасности, а именно:

- быть `SYSDBA`;
- иметь 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 - быть администратором 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 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

Представление (view) — это виртуальная таблица, значения которой в базе данных не хранится. Основой представления является оператор выборки SELECT произвольной сложности, который задает выборку данных из одной или более таблиц базы данных. В базе данных хранится именно этот оператор SELECT, но не результаты его выполнения. Результат в виде набора данных динамически создается при обращении к представлению.

Оператор CREATE VIEW создает новое представление. Его синтаксис:

**Листинг Д.48. Синтаксис оператора CREATE VIEW**

```
CREATE VIEW <имя представления>
   [( <имя столбца> [AS <псевдоним>] [, <имя столбца> [AS <псевдоним>] ...] )]
AS <оператор SELECT>
[WITH CHECK OPTION];
```

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.

После имени создаваемого представления может идти список столбцов исходных таблиц, получаемых в результате обращения к представлению. 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.

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. В операторе нельзя указывать предложение ORDER BY и как следствие этого предложение ROWS. Использование предложений FIRST и SKIP допустимо. Оператор может обращаться только к таблицам базы данных или к другим представлениям. Нельзя в этом операторе обращаться к хранимым процедурам. Основные характеристики представления определяются главным (первым) оператором SELECT, лежащим в основе данного 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 обычны-

ми 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 создания изменяемых представлений из неизменяемых см. в главе 9 «Работа с представлениями».

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, допускают пустые значения NULL;
- 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.);
- 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.

Необязательное предложение WITH CHECK OPTION задает для изменяемого представления требование проверки соответствия вновь вводимых или изменяемых данных условию, заданному в предложении WHERE. При попытке поместить новую строку, которая не соответствует условию выборки в предложении WHERE, 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 главном операторе SELECT представления указано предложение WHERE. Иначе будет выдано сообщение об ошибке.

Подробно оператор создания представления, а также список системных представлений описаны в главе 9 «Работа с представлениями».

Возможности оператора SELECT см. в разделе 8.1 «SELECT».

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 Д.49. 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>';
```

Имя UDF — 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 ключевым словом задается максимальное число символов, которое может присутствовать в строке, включая завершающее нулевое значение.

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.

Ключевое слово BY VALUE означает, что данное 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.

Объявить внешнюю функцию может пользователь с административными привилегиями и пользователь с привилегией CREATE FUNCTION. Пользователь, объявивший внешнюю функцию, становится её владельцем.

Список функций, определенных пользователем, представлен в приложении Г «Функции, определенные пользователем (UDF)».

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 Д.50. 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
```

Имя BLOB фильтра должно быть уникальным среди имен BLOB фильтров.

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 вход-

ных 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.

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.

Создать BLOB фильтр может администратор и пользователь с привилегией CREATE FILTER.

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 Д.51. 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 <значение 1> [TO <значение 2>]]
[RETURNING <имя столбца> [[AS] <алиас>] [, <имя столбца> [[AS] <алиас>]...]
   [INTO [:]<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 определяет множество строк, которые будут удалены. Если это предложение не указано, то будут удалены все существующие строки таблицы (таблиц, водящих в состав представлении) в том случае, если не указано также предложение ORDER BY и предложение ROWS.

Подробнее об условиях поиска в предложении WHERE см. в разделе 8.1 «SELECT».

В операторе может быть использовано ключевое слово PLAN, задающее план выборки данных для удаления. Подробнее о плане выборки см. в разделе 8.1 «SELECT». Основным назначение этого предложения является определение порядка выборки строк из исходной таблицы (таблиц) для выполнения удалений.

Предложение ORDER BY следует использовать, если далее будет задано предложение ROWS. Предложение ORDER BY используется для упорядочения результатов выборки. В нем указывает-

ся список столбцов, по которым происходит упорядочение, направление сортировки для каждого столбца (по возрастанию или по убыванию) и порядок сортировки (COLLATE) для строкового столбца. Синтаксис для предложения ORDER BY:

```
ORDER BY <упорядочиваемый элемент> [, <упорядочиваемый элемент> ... ]

<упорядочиваемый элемент> ::=
   {<имя столбца>|<псевдоним столбца>|<номер столбца>|<произвольное выражение>}
   [COLLATE <порядок сортировки>]
   [ASC[ENDING] | DESC[ENDING]]
   [NULLS {FIRST | LAST}].
```

В предложении перечисляются столбцы, по которым нужно упорядочить строки набора данных перед выполнением удаления. Можно задавать только имена столбцов, номера недопустимы.

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 одним столбцам может идти по возрастанию значений, а по другим — по убыванию.

Ключевое слово COLLATE позволяет задать порядок сортировки строкового столбца, если нужен порядок сортировки, отличный от того, который был установлен для этого столбца (явно или по умолчанию). Допустимые порядки сортировки для различных наборов символов см. в приложении В «Наборы символов и порядок сортировки».

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.

Необязательное предложение ROWS задает диапазон строк, к которым будет применена операция удаления данных. Предложение ROWS можно использовать, только если задано предложение ORDER BY.

```
ROWS <значение 1> [TO <значение 2>]
```

Значение 1 задает количество включаемых в операцию удаления строк, упорядоченных в предложении ORDER BY, если не задан вариант TO. Это первые строки в упорядоченном списке.

Значение 1 задает начальный номер удаляемой строки в упорядоченном списке строк, если задан вариант TO. Значение 2 в этом случае указывает конечный номер удаляемой строки.

Предложение RETURNING позволяет вернуть вызвавшей программе значения указанных столбцов удаленной строки. Если происходит удаление более чем одной строки, то выдается сообщение об ошибке базы данных. Ключевое слово INTO позволяет сохранить возвращенные значения во внутренних переменных триггера или хранимой процедуры. Вариант INTO может использоваться только в PSQL — см. главу 11 «Хранимые процедуры и триггеры».

Подробнее об операторе удаления см. вглаве 8 «Операторы DML».

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 Д.52. 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 Д.53. 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 3 «Работа с базой данных».

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 Д.54. 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 Д.55. 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 Д.56. 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 Д.57. 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 GENERATOR

Оператор DROP GENERATOR (DROP SEQUENCE) удаляет генератор из базы данных. Синтаксис:

**Листинг Д.58. 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 6 «Работа с генераторами».

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 Д.59. 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 7 «Работа с индексами».

См. также операторы CREATE INDEX, ALTER INDEX.

# DROP FUNCTION

Для удаления существующей хранимой функции используется оператор DROP FUNCTION. Синтаксис оператора представлен в листинге Д.60.

**Листинг Д.60. 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 MAPPING

Оператор DROP MAPPING удаляет существующее отображение. Если указана опция GLOBAL, то будет удалено глобальное отображение.

**Листинг Д.61. 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 представлен в листинге Д.62.

**Листинг Д.62. 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.

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 представлен в листинге Д.63.

**Листинг Д.63. 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.

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>
```

Возможные параметры политик, а также у каких пользователей есть права на операцию DROP POLICY можно посмотреть в описании параметра CREATE POLICY.

См. также операторы CREATE POLICY, ALTER POLICY.

# DROP PROCEDURE

Для удаления существующей хранимой процедуры используется оператор DROP PROCEDURE. Синтаксис оператора:

**Листинг Д.64. 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.

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 Д.65. 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 Д.66. 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 6 «Работа с генераторами».

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 Д.67. 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 3 «Работа с базой данных».

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 Д.68. 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 5 «Работа с таблицами».

См. также операторы CREATE TABLE, ALTER TABLE.

# DROP TRIGGER

Оператор DROP TRIGGER удаляет существующий триггер. Синтаксис оператора:

**Листинг Д.69. 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.

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 Д.70. 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 Д.71. 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.

См. также операторы CREATE VIEW, RECREATE VIEW, CREATE OR ALTER VIEW, ALTER VIEW.

# EXECUTE BLOCK

Оператор EXECUTE BLOCK позволяет из декларативной части SQL (из ядра языка манипулирования данными DML) выполнять действия, доступные в хранимых процедурах и триггерах.

**Листинг Д.72. 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 процедуры>
                 | <объявление 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.

В блоке операторов выполняются произвольные действия по обработке данных.

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 Д.73. 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.

Если этот оператор вызывается из isql, то нельзя использовать предложение RETURNING_VALUES.

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 Д.74. 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 <пользователь получатель> {,<пользователь получатель>}

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
  | VIEW
  | PROCEDURE
  | FUNCTION
  | PACKAGE
  | GENERATOR
  | SEQUENCE
  | DOMAIN
  | EXCEPTION
  | ROLE
  | CHARACTER SET
  | COLLATION
  | FILTER }

<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> }

<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.

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.

При предоставлении привилегий пользователям можно указать предложение WITH GRANT OPTION, что позволяет в свою очередь предоставлять другим пользователям эти привилегии.

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 только 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.

В Ред Базе Данных 3.0 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>
```

См. также операторы CREATE ROLE, DROP ROLE, REVOKE, CONNECT.

# INSERT

Оператор INSERT добавляет в таблицу или в представление одну или более строк. Синтаксис оператора:

**Листинг Д.75. Синтаксис оператора INSERT**

```
INSERT INTO {<имя таблицы> | <имя представления>} {
    [(<список столбцов>)] {VALUES (<значение> [,<значение> ...])|<поиск многих>}
  | DEFAULT VALUES }
[RETURNING <имя столбца> [[AS] <алиас>] [, <имя столбца> [[AS] <алиас>]...]
   [INTO [:]<имя переменной> [, [:]<имя переменной> ...]] ];

<список столбцов> ::= <имя столбца> [, <имя столбца> ...]
```

Оператор добавляет строки в обычную таблицу или в таблицу изменяемого представления — см. главу 9 «Работа с представлениями».

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 — см. документ «Руководство администратора».

Можно добавить строку, для которой указаны конкретные значения столбцов, или строку, которая будет во всех столбцах содержать значения по умолчанию (предложение DEFAULT VALUES).

Список столбцов, в которые помещаются данные, заключен в круглые скобки. Если какой-либо столбец не задан в списке, то ему присваивается пустое значение NULL или значение по умолчанию, если оно задано для столбца (предложение DEFAULT в определении столбца).

Необязательное предложение RETURNING указывает, что оператор возвращает значения заданных столбцов добавляемой строки. Если оператор помещает более одной строки в таблицу, то в этом случае возникнет ошибка базы данных. Ключевое слово INTO позволяет сохранить эти возвращенные значения во внутренних переменных триггера или хранимой процедуры.

Синтаксис «значения», используемого в операторе:

```
<значение> ::= {
    <литерал>
  | <выражение>
  | <встроенная функция>
  | <UDF> [(<параметр> [, <параметр> ...])]
  | <обращение к хранимой процедуре> [(<параметр> [, <параметр>]...)]
  | NEXT VALUE FOR <имя генератора>
  | (<выбор одного>) }
```

Чаще всего значениями являются литералы, то есть самоопределенные константы, предварительно определенные литералы, контекстные переменные.

В SQL Ред База Данных существует два типа встроенных функций — обычные встроенные функции и агрегатные функции в операторе SELECT.

Обычная встроенная функция — это функция, получающая один или более параметров, которая не связана с оператором SQL выборки данных SELECT. Функция возвращает ровно одно значение. Параметры передаются таким функциям на основании принятого для каждой функции синтаксиса. Описание встроенных функций см. в Приложении Е.

Агрегатная функция в операторе SELECT определяется следующим образом:

**Листинг Д.76. агрегатная функция в операторе SELECT**

```
<агрегатная функция в операторе SELECT> ::=
   SELECT {
       COUNT ({[ALL | DISTINCT] <выражение> | *})
```

```
      | SUM ([ALL | DISTINCT] <выражение>)
      | AVG ([ALL | DISTINCT] <выражение>)
      | MAX ([ALL | DISTINCT] <выражение>)
      | MIN ([ALL | DISTINCT] <выражение>)
      | LIST ([ALL | DISTINCT] <выражение>) [, '<разделитель>']
      | CORR (<выражение1>, <выражение2>)
      | COVAR_POP (<выражение1>, <выражение2>)
      | COVAR_SAMP (<выражение1>, <выражение2>)
      | STDDEV_POP (<выражение>)
      | STDDEV_SAMP (<выражение>)
      | VAR_POP (<выражение>)
      | VAR_SAMP (<выражение>)
      | REGR_AVGX (y, x)
      | REGR_AVGY (y, x)
      | REGR_COUNT (y, x)
      | REGR_INTERCEPT (y, x)
      | REGR_R2 (y, x)
      | REGR_SLOPE (y, x)
      | REGR_SXX (y, x)
      | REGR_SXY (y, x)
      | REGR_SYY(y, x) }
    <предложение FROM>
    [<предложение WHERE>]
```

Конструкция `NEXT VALUE FOR` используется вместо функции `GEN_ID`.

Здесь также может быть использован оператор `SELECT` (в синтаксисе оператора задано конструкцией «выбор одного»), возвращающий ровно одно значение на основании условий поиска в таблице или в группе таблиц. При использовании оператора `SELECT` пользователь должен иметь соответствующие полномочия к данным таблицам.

Подробнее об операторе добавления см. в главе 8 «Операторы DML».

См. также операторы `UPDATE`, `UPDATE OR INSERT`, `DELETE`, `SELECT`, `SET TRANSACTION`, функции `AVG()`, `SUM()`, `MIN()`, `MAX()`, `COUNT()`, `CAST()`, `GEN_ID()`, `NEXT VALUE FOR`.

# MERGE

Объединяет данные в таблицу или представление.

**Листинг Д.77. Синтаксис оператора `MERGE`**

```
MERGE INTO <имя таблицы | имя представления> [[AS] <алиас>]
USING <таблица|представление|хранимая процедура|(оператор Select)> [AS <алиас>]
ON <условие соединения>
<предложение WHEN> [<предложение WHEN> ...]
[RETURNING <список возвращаемых выражений> [INTO <список переменных>]]

<предложение WHEN> ::= <предложение WHEN MATCHED> | <предложение WHEN NOT MATCHED>

<предложение WHEN MATCHED> ::=
  WHEN MATCHED [ AND <доп.условие> ]
  THEN {UPDATE SET <имя столбца>=<значение>[,<имя столбца>=<значение>...] | DELETE}

<предложение WHEN NOT MATCHED> ::=
  WHEN NOT MATCHED [ AND <доп.условие> ]
  THEN INSERT [(<столбцы>)] VALUES (<значения>)

<столбцы> ::= <имя столбца> [, <имя столбца> ...]
```

```
<значения> ::= <значение> [, <значение> ...]
<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 или Common Table Expressions. 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.

Допускается указывать несколько предложений WHEN MATCHED и WHEN NOT MATCHED.

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 стандар-

ту: SQL-2003 определяет, что в таком случае должно быть вызвано исключение .

# RECREATE EXCEPTION

Оператор создает новое пользовательское исключение, если оно отсутствует в базе данных, или пересоздает существующее. Синтаксис оператора:

**Листинг Д.78. Синтаксис оператора RECREATE EXCEPTION**

```
RECREATE EXCEPTION <имя исключения> '<текст сообщения>';
```

Имя исключения может содержать до 31 символа 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 представлен в листинге Д.79.

**Листинг Д.79. 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
   [<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 представлен в листинге Д.80.

**Листинг Д.80. 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> ::=
   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 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 представлен в листинге Д.81.

**Листинг Д.81. 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
         [<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
         [<объявление> [<объявление> ...] ]
       BEGIN
         <блок операторов>
       END }

<входной параметр> ::= <описание параметра> [{=|DEFAULT} <значение по умолчанию>]
<входной_параметр> ::= <описание параметра>
<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>
               | <объявление функции>
```

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 Д.82. 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
   [<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>
                | <объявление 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 Д.83. 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 Д.84. Синтаксис оператора создания таблицы RECREATE TABLE**

```
RECREATE TABLE <имя таблицы>
   [EXTERNAL [FILE] '<спецификация файла>']
   (<определение столбца> [, {<определение столбца> | <ограничение таблицы>}...])
   [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 Д.85. 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
   [<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
 | 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>
                 | <объявление 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 VIEW

Оператор `RECREATE VIEW` позволяет внести изменения в существующее представление. Его синтаксис:

**Листинг Д.86. Синтаксис оператора `RECREATE VIEW`**

```
RECREATE VIEW <имя представления>
```

```
   [( <имя столбца> [AS <псевдоним>] [, <имя столбца> [AS <псевдоним>] ...] )]
AS <оператор SELECT>
[WITH CHECK OPTION];
```

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 Д.87. 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 10 «Транзакции».

См. также операторы SET TRANSACTION, ROLLBACK, SAVEPOINT, COMMIT.

# RESET USER

Оператор позволяет разблокировать заблокированного многофакторного пользователя. Синтаксис оператора RESET USER:

**Листинг Д.88. Синтаксис оператора RESET USER**

```
RESET USER <имя пользователя>
```

Для разблокировки учетной записи многофакторного пользователя текущий пользователь должен обладать административными привилегиями в базе данных безопасности

См. также операторы CREATE USER, DROP USER.

# 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 Д.89. 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
  | VIEW
  | PROCEDURE
  | FUNCTION
  | PACKAGE
  | GENERATOR
  | SEQUENCE
  | DOMAIN
  | EXCEPTION
  | ROLE
  | CHARACTER SET
  | COLLATION
  | FILTER }

<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> }

<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.

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.

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.

Предложение `ADMIN OPTION FOR` отменяет ранее предоставленную опцию — право на передачу предоставленной пользователю роли другим, не отменяя прав на роль.

Используя предложение `GRANTED BY` можно отозвать привилегии не от имени текущего пользователя, а от другого пользователя. Данное предложение доступно не всем пользователям (см. `GRANT`).

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. `GRANT`.

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 Д.90. Синтаксис оператора ROLLBACK**

```
ROLLBACK [WORK] [TRANSACTION <имя транзакции>]
[RETAIN [SNAPSHOT] | TO SAVEPOINT <имя точки сохранения>] [RELEASE];
```

Необязательное предложение TRANSACTION указывает, действия какой именно транзакции отменяются. Если имя не указано, то предполагается транзакция по умолчанию с именем GDS_TRANS.

Необязательное предложение TO SAVEPOINT задает имя точки сохранения, на которую происходит откат. Контрольная точка с этим именем должна быть создана в контексте транзакции. Все точки сохранения, созданные после точки сохранения, на которую производится откат, освобождаются. Если точки сохранения с этим именем в списке не существует, то не выполняется никаких действий.

Необязательное ключевое слово WORK используется для совместимости с другими системами управления реляционными базами данных.

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 последнему выполнению оператора ROLLBACK RETAIN. В случае отмены транзакции с сохранением ее контекста нет необходимости заново выполнять оператор SELECT для получения данных из таблицы.

Подробно оператор, работа с транзакциями, использование вложенных транзакций описаны в главе 10 «Транзакции».

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 Д.91. Синтаксис оператора создания точки сохранения SAVEPOINT**

```
SAVEPOINT <имя точки сохранения>;
```

Имя точки сохранения — идентификатор базы данных, который может содержать до 31 символа. 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 10 «Транзакции».

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 Д.92. Синтаксис оператора SELECT**

```
[WITH [RECURSIVE] <CTE> [, <CTE> ...]]
SELECT
  [FIRST <значение>] [SKIP <значение>]
  [DISTINCT | ALL]
  <выходное поле> [, <выходное поле>]
FROM
  <источники>
  [<соединяемые источники>]
[WHERE <условие выборки>]
[GROUP BY <условие группирование выбранных данных>
 [HAVING <условие выборки>]]
[UNION [DISTINCT | ALL] <другой набор данных>]
[PLAN <выражение для плана поиска>]
[ORDER BY <выражение для порядка выборки>]
[OPTIMIZE FOR {FIRST | ALL} ROWS]
[   ROWS <m> [TO <n>]
  | [OFFSET <n> {ROW | ROWS}] [FETCH {FIRST | NEXT} [<m>] {ROW | ROWS} ONLY] ]
[FOR UPDATE [OF <имя столбца> [, <имя столбца>]...]]
[WITH LOCK]
[INTO [:]<переменная> [,[:]<переменная> ... ]]
```

Предложение WITH позволяет задать общее выражение таблицы (CTE, Common Table Expression). Оно может быть рекурсивным (ключевое слово RECURSIVE) и обычным, не рекурсивным (значение по умолчанию).

Сразу после предложения SELECT могут следовать предложения FIRST и SKIP, позволяющие определить количество помещаемых в результирующий набор данных строк, выбранных на основании условий выборки (предложения ON, UNION и WHERE). Ключевое слово DISTINCT указывает, что в выходной набор данных не помещаются дубликаты строк. Далее идет список выбора, указывающий, какие столбцы из каких таблиц, участвующих в операции выборки (объединения, соединения), 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 внешнего или внутреннего соединения (JOIN) нескольких таблиц для получения выходного набора 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 (см. Приложение Е).

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 10 «Транзакции».

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).

## Список выбора

После ключевого слова SELECT может следовать указание (предложения FIRST и SKIP), какое количество полученных при выполнении этого оператора строк исходной таблицы (исходных таблиц) должно помещаться в выходной набор данных. Синтаксис предложений FIRST и SKIP:

```
SELECT [FIRST <значение 1>] [SKIP <значение 2>]
```

Предложение FIRST указывает, что в выходной набор данных должно быть помещено заданное количество первых выбранных строк (значение 1). Это предложение может использоваться самостоятельно или вместе с предложением SKIP.

В случае предложения SKIP указывается, что заданное количество первых строк не должно помещаться в выходной набор данных. Это предложение может использоваться самостоятельно или вместе с ключевым словом FIRST. Тогда из множества строк, выбранных по варианту FIRST, удаляются строки в количестве, указанном в предложении SKIP.

Количество помещаемых в результирующий набор данных строк может также задаваться при использовании предложений ORDER BY и ROWS, а также с помощью предложений OFFSET, FETCH.

Выбираемые строки оператором SELECT определяются условием в предложении WHERE и в предложениях ON, если в операторе используется соединение таблиц. В выходной набор данных могут попасть не все эти строки, а меньшее их количество. Для уменьшения объема выборки используются либо ключевые слова FIRST и/или SKIP, либо предложение ORDER BY вместе с предложением ROWS, либо предложения OFFSET, FETCH.

Предложение ROWS задает диапазон строк, которые попадут в результирующий набор данных из полученного в результате выполнения запроса набора данных. Предложение ROWS можно использовать, если задано и предложение ORDER BY.

В одном предложении SELECT не могут одновременно присутствовать ключевые слова FIRST и SKIP и в то же время предложение ROWS и OFFSET, FETCH. В любом случае в операторе может присутствовать предложение WHERE.

Перед списком выбора может находиться ключевое слово ALL или DISTINCT.

Ключевое слово ALL (это значение по умолчанию) указывает, что в выходной набор данных должны помещаться все строки, соответствующие условию поиска.

Ключевое слово DISTINCT не позволяет помещать в выходной набор данных дубликаты строк.

Сам список выбора, определяющий, какие столбцы из каких таблиц должны помещаться в результирующий набор данных, помещается сразу после ключевого слова ALL или DISTINCT.

Символ «*» в списке выбора означает, что в результирующем наборе данных должны присутствовать все столбцы исходной таблицы (исходных таблиц).

В списке могут присутствовать имена столбцов, константы, выражения. После ключевого слова AS может следовать псевдоним столбца. Этот псевдоним становится именем столбца в выходном наборе данных.

Если в запросе используется несколько таблиц, у которых имена столбцов могут совпадать, то слева от имени нужного столбца прибавляется имя таблицы или псевдоним таблицы и точка. Такая конструкция называется уточненным именем столбца:

```
[{<имя таблицы> | <псевдоним таблицы>}.]<имя столбца>
```

Если для таблицы был указан псевдоним, то везде в операторе следует использовать только псевдоним, а не имя таблицы.

## Предложение FROM

Предложение FROM задает таблицу или список таблиц, из которых осуществляется выборка данных. Могут также указываться представления и хранимые процедуры выбора. Для хранимой процедуры можно указать список передаваемых процедуре входных параметров. Если в предложении задан список таблиц, то это соответствует неявному внутреннему соединению (INNER JOIN). Здесь условие соединения задается в предложении WHERE.

```
<источники> ::= {
    <таблица>
  | <представление>
  | <селективная хранимая процедура> [(<аргументы>)]
  | <производная таблица>
  | <псевдоним CTE>
} [[AS] <псевдоним>]

<производная таблица> ::= (<SELECT запрос>) [[AS] <псевдоним производной таблицы>]
[(<псевдоним столбца производной таблицы> [, <псевдоним столбца>])]
```

После имени таблицы (представления, процедуры выбора) может следовать псевдоним этой таблицы. Псевдоним таблицы (объекта выборки), если он указан, должен быть использован во всех дальнейших конструкциях при обращении к таблице (к объекту выборки) или к ее столбцам. В этом случае уже недопустимо использование имени таблицы.

```
<соединяемые источники> ::= <соединение> [ <соединение> ...]

<соединение> ::=
  [ <вид соединения>] JOIN <источники> <условие соединения>
  | NATURAL [<вид соединения>] JOIN <источники>
  | {CROSS JOIN | ,} <источники>

<вид соединения> ::= INNER | {LEFT | RIGHT | FULL} [OUTER]

<условие соединения> ::= ON <условие> | USING (<список столбцов>)
```

Соединение может быть внутренним (ключевое слово INNER можно опустить) или внешним (OUTER). Внешнее соединение бывает левым (LEFT), правым (RIGHT) и полным (FULL).

Условие соединения для строк исходных таблиц задается после ключевого слова ON. Как правило, это проверка на равенство значений столбцов из двух соединяемых таблиц.

Для внутреннего (INNER) соединения вид соединения можно не указывать — соединение является внутренним по умолчанию. Результатом внутреннего соединения двух таблиц является декартово произведение двух множеств (строк таблиц). В выходной набор данных попадают только те строки полученного декартового произведения, которые соответствуют условию соединения, заданному в предложении ON.

Внешние соединения (OUTER JOIN) бывают левыми (LEFT), правыми (RIGHT) и полными (FULL).

При левом внешнем соединении (LEFT OUTER JOIN) к строкам первой, левой, таблицы добавляются данные из второй, правой, присоединяемой, таблицы на основании условий соединения, заданных в предложении ON. Если в правой таблице нет соответствующей строки, то столбец первой соединяемой таблицы будет иметь пустое значение NULL. Общее количество строк, попадающих в результирующий набор данных, определяется условием в предложении WHERE.

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 внешнем соединении — справа налево.

В случае полного внешнего соединения (FULL OUTER JOIN) выбираются все, соответствующие условию в предложении WHERE строки как левой, так и правой таблицы. Затем между этими строками устанавливается соответствие, заданное в предложении ON.

## Предложение WHERE

Предложение WHERE определяет множество строк, которые будут выбираться из исходных таблиц. Если это предложение не указано, будут выбраны все существующие строки таблицы в том случае, если не указано также и предложение ORDER BY и предложение ROWS.

Синтаксис условия выборки данных:

```
<условие выборки> ::= {
    <значение> <оператор сравнения> { <значение 1> | (<выбор одного>) }
  | <значение> [NOT] IN ({ <значение 1> [, <значение 2>]... | <поиск одного> })
  | <значение> [NOT] BETWEEN <значение 1> AND <значение 2>
  | <значение> [NOT] LIKE <значение 1> [ESCAPE '<символ>']
  | <значение> IS [NOT] NULL
  | <значение> IS [NOT] DISTINCT FROM <значение 1>
  | <значение> <оператор сравнения> { ALL | SOME | ANY} ( <поиск одного> )
  | EXISTS (<поиск многих>)
  | SINGULAR (<поиск многих>)
  | <значение> [NOT] CONTAINING <значение 1>
  | <значение> [NOT] STARTING [WITH] <значение 1>
  | (<условие выборки>)
  | NOT <условие выборки>
  | <условие выборки> OR <условие выборки>
  | <условие выборки> AND <условие выборки> }
```

Оператором в этом условии является оператор сравнения.

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

Он 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.

Синтаксис значения в условии выборки данных определяется следующим образом:

```
<значение> ::= {
 [{ <имя таблицы> | <псевдоним таблицы> }.]<имя столбца> [[<элемент массива>]]
  | <литерал>
  | <выражение>
  | NEXT VALUE FOR <имя генератора>
  | <обычная внутренняя функция> (<параметры>)
  | <агрегатная функция в операторе SELECT>
  | <функция UDF> [(<параметр> [, <параметр>]...)]
  | NULL }
```

Можно указать имя столбца таблицы. Если несколько таблиц в операторе SELECT имеют столбцы с одинаковыми именами, то во избежание двусмысленности следует указать перед именем столбца имя или псевдоним таблицы и точку. Если для таблицы в операторе задан псевдоним, то можно указывать только псевдоним, но не имя таблицы.

Литерал — это числовая константа, строковая константа, заключенная в апострофы, литерал даты или времени, предварительно определенный литерал, контекстная переменная (см. главу 2 «Типы данных Ред База Данных»).

Обычная внутренняя функция — это функция, работающая с одним или более параметрами, которая не связана с оператором SQL выборки данных SELECT. Функция возвращает ровно одно значение. Синтаксис обычной внутренней функции:

```
<обычная внутренняя функция> ::= {
    CAST ( <значение> AS <тип данных> [CHARACTER SET <набор символов>] )
  | UPPER (<строковое значение>)
  | LOWER (<строковое значение>)
  | TRIM ([[<спец-ия удаления>][<удаляемые символы>] FROM] <строковое значение>)
  | { CHARACTER_LENGTH | CHAR_LENGTH } (<параметр функции>)
  | OCTET_LENGTH (<параметр функции>)
  | BIT_LENGTH (<параметр функции>)
  | SUBSTRING (<строка> FROM <начальная позиция> [FOR <длина подстроки>]
            | <строковое значение> SIMILAR <шаблон> ESCAPE <символ экранирования>)
  | EXTRACT (<выделяемый элемент> FROM <исходное данное>)
  | GEN_ID (<генератор>, <приращение>)
  | CASE <выражение 1>
    WHEN <выражение 2> THEN {<выражение 3> | NULL}
   [WHEN <выражение 2> THEN {<выражение 3> | NULL}] ...
   [ELSE {<выражение 4> | NULL}]
    END
  | COALESCE (<выражение> [, <выражение>] ...)
  | NULLIF(<значение 1>, <значение 2>)
  | IIF (<условие>, <значение 1>, <значение 2>)
}
<спецификация удаления> ::= LEADING | TRAILING | BOTH
```

Агрегатные функции в операторе SELECT — функции, определенные в языке SQL Ред База Данных. Они работают не с одним фиксированным набором параметров, а с группой значений, полученных при выполнении оператора SELECT из таблицы базы данных. Агрегатные функции используются внутри списка выбора оператора SELECT. Синтаксис агрегатной функции в операторе SELECT:

```
<агрегатная функция в операторе SELECT> ::=
   SELECT {
       COUNT ({[ALL | DISTINCT] <выражение> | *})
     | SUM ([ALL | DISTINCT] <выражение>)
     | AVG ([ALL | DISTINCT] <выражение>)
     | MAX ([ALL | DISTINCT] <выражение>)
     | MIN ([ALL | DISTINCT] <выражение>)
     | LIST ([ALL | DISTINCT] <выражение>) [, '<разделитель>']
     | CORR (<выражение1>, <выражение2>)
     | COVAR_POP (<выражение1>, <выражение2>)
     | COVAR_SAMP (<выражение1>, <выражение2>)
     | STDDEV_POP (<выражение>)
     | STDDEV_SAMP (<выражение>)
     | VAR_POP (<выражение>)
     | VAR_SAMP (<выражение>)
     | REGR_AVGX (y, x)
     | REGR_AVGY (y, x)
     | REGR_COUNT (y, x)
     | REGR_INTERCEPT (y, x)
     | REGR_R2 (y, x)
```

```
      | REGR_SLOPE (y, x)
      | REGR_SXX (y, x)
      | REGR_SXY (y, x)
      | REGR_SYY(y, x) }
    <предложение FROM>
    [<предложение WHERE>]
```

Конструкция NEXT VALUE FOR используется вместо функции GEN_ID.

Оператор IN указывает, что значение в столбце выбираемой таблицы должно находиться (или не находиться, если указано ключевое слово NOT) в заданном списке.

В операторе BETWEEN проверяется присутствие значения, записанного в левой части условия, в диапазоне, заданном в правой части условия, включая граничные значения.

Оператор LIKE задает проверку наличия (или отсутствия в случае указания необязательного ключевого слова NOT) в значении столбца символьного типа данных определенных символов.

Оператор IS NULL осуществляет проверку на пустое значение.

Оператор IS DISTINCT FROM выполняет проверку на равенство (неравенство, если задано NOT) двух значений.

Функция ALL возвращает значение «истина», если сравнение будет истинным для всех значений столбца, полученных из оператора SELECT.

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

Аргументом функции EXISTS является оператор SELECT, возвращающий произвольное количество любых столбцов таблицы. Результатом будет «истина», если оператор SELECT вернет хотя бы одно значение, соответствующее условиям поиска, заданным в предложении WHERE.

Аргументов функции SINGULAR является оператор SELECT, возвращающий произвольное количество любых столбцов таблицы. Результатом будет «истина», если оператор SELECT вернет в точности одно значение, соответствующее условиям поиска, заданным в предложении WHERE.

Результатом выполнения оператора CONTAINING будет «истина», если значение в левой части выражения будет содержать в качестве своей части указанное значение. Этот оператор не чувствителен к регистру.

Результатом выполнения оператора STARTING WITH будет «истина», если значение в левой части выражения будет начинаться с символов, указанных в правой части. Оператор чувствителен к регистру, однако такое ограничение можно обойти, используя функцию UPPER.

## Предложение GROUP BY и HAVING

Необязательное предложение GROUP BY задает условие группирования выбранных данных в соответствии со значением указанного столбца (указанных столбцов). 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 сокращает количество отобранных строк в результирующем выходном наборе данных. В предложении GROUP BY можно использовать имена столбцов, их псевдонимы или номера столбцов в заданном в операторе SELECT списке выбора.

Синтаксис предложения GROUP BY:

```
GROUP BY {
    <имя столбца из списка выбора>
  | <псевдоним столбца из списка выбора>
  | <номер столбца в списке выбора>
  | <не агрегатное выражение, не включено в список выборки>}
  [, { <имя столбца из списка выбора>
     | <псевдоним столбца из списка выбора>
     | <номер столбца в списке выбора>
```

```
    | <не агрегатное выражение, не включено в список выборки>} ...]
[HAVING <условие выборки>]
```

Предложение GROUP BY позволяет сгруппировать полученные в результате выполнения оператора SELECT строки. 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.

## Предложение UNION

Предложение UNION позволяет объединить с выбранным набором данных другой набор данных, имеющий точно такую же структуру. Синтаксис предложения объединения:

```
UNION [DISTINCT | ALL] <другой набор данных>
```

Присоединяемый набор данных должен иметь тот же состав столбцов по количеству и типам данных, что и основной набор данных, полученный главным оператором SELECT. Для строковых типов данных допустимы отличающиеся размеры.

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.

Другой набор данных, указанный в синтаксисе оператора UNION, — это оператор SELECT, который обращается к другой или той же самой объединяемой таблице.

В одном операторе SELECT может присутствовать более одного объединения.

## Предложение PLAN

В операторе можно задать свой план выборки данных при помощи предложения PLAN.

```
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> ...])

<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> ...]) ] }
```

Выражение для плана поиска допускает вложенные конструкции.

Если выполняется запрос к нескольким таблицам, то на основании плана осуществляется выборка данных из каждой таблицы. Результатом одной такой выборки является промежуточный набор данных. На следующем этапе выполняется соединение (JOIN) или объединение, слияние (MERGE) промежуточных наборов данных в один результирующий набор данных, который и является результатом выполнения оператора SELECT.

В выражении плана в самом начале может присутствовать тип соединения.

JOIN — тип соединения по умолчанию. В этом случае с левым 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 в первую очередь.

## Предложение ORDER BY, ROWS, OFFSET, FETCH

Данные, полученные из оператора SELECT, никак не упорядочены. Предложение ORDER BY позволяет упорядочить результирующий набор данных.

```
ORDER BY <упорядочиваемый элемент> [, <упорядочиваемый элемент> ... ] [   ROWS
<значение1> [TO <значение2>]
  | [OFFSET <n> {ROW | ROWS}] [FETCH {FIRST | NEXT} [<m>] {ROW | ROWS} ONLY] ]

<упорядочиваемый элемент> ::=
   {<имя столбца>|<псевдоним столбца>|<номер столбца>|<произвольное выражение>}
   [COLLATE <порядок сортировки>]
   [ASC[ENDING] | DESC[ENDING]]
   [NULLS {FIRST | LAST}]
```

В предложении через запятую перечисляются столбцы, по которым нужно упорядочить результирующий набор данных. Можно задавать имя столбца, псевдоним, присвоенный столбцу в списке выбора при помощи ключевого слова AS, или порядковый номер столбца в списке выбора. В одном предложении можно для разных столбцов смешивать форму записи. Например, один столбец может быть задан своим именем, а другой порядковым номером.

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 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 приложении В «Наборы символов и порядок сортировки».

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.

Необязательное предложение ROWS задает диапазон строк, которые попадут в результирующий набор данных. Предложение ROWS можно использовать, только если задано предложение ORDER BY.

Значение 1 задает количество включаемых в выходной набор данных строк, упорядоченных в предложении ORDER BY, если не задан вариант TO. Это первые строки в упорядоченном по ORDER BY списке.

Значение 1 задает начальный номер строки в упорядоченном списке строк, если задан вариант TO. Значение 2 в этом случае указывает конечный номер строки.

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.

Предложения OFFSET и/или FETCH не могут быть объединены с предложениями ROWS или FIRST/SKIP в одном выражении запроса.

## Предложение FOR UPDATE, WITH LOCK и INTO

Необязательное предложение FOR UPDATE означает, что полученный набор не весь сразу передается клиенту, а по отдельным строкам на основании запросов клиента. Необязательное предложение WITH LOCK запрещает параллельным процессам, транзакциям выполнять какие-либо изменения в данной таблице запроса. Подробнее см. в главе 10 «Транзакции».

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 8.1 «SELECT».

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 GENERATOR

Оператор позволяет задать новое значение для генератора. Синтаксис:

**Листинг Д.93. 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 6 «Работа с генераторами».

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 Д.94. 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 приложении В «Наборы символов и порядок сортировки».

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 Д.95. Синтаксис оператора SET ROLE**

```
SET ROLE <имя роли>;
```

См. также оператор SET TRUSTED ROLE.

# SET SQL DIALECT

Оператор задает диалект клиента для выполнения доступа к базе данных.

**Листинг Д.96. 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 STATISTICS

Оператор позволяет улучшить селективность (избирательность) указанного индекса. Улучшенная селективность увеличивает скорость выборки и упорядочения данных, при которых используется данный индекс. Синтаксис оператора:

**Листинг Д.97. 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 7 «Работа с индексами».

См. также операторы CREATE INDEX, DROP INDEX, ALTER INDEX.

# SET TRANSACTION

Оператор запускает на выполнение транзакцию с указанным именем и с заданными характеристиками. Синтаксис оператора:

**Листинг Д.98. Синтаксис оператора SET TRANSACTION**

```
SET TRANSACTION
[NAME <имя транзакции>]
[READ WRITE | READ ONLY]
[WAIT [LOCK TIMEOUT <кол-во секунд>] | NO WAIT]
[[ISOLATION LEVEL]
   { SNAPSHOT [TABLE STABILITY] | READ COMMITTED [[NO] RECORD_VERSION] } ]
[NO AUTO UNDO]
[IGNORE LIMBO]
[RESERVING <предложение резервирования> | USING <хендл базы данных>];

<предложение резервирования> ::=
   <имя таблицы> [, <имя таблицы> ...]
   [FOR [SHARED | PROTECTED] {READ | WRITE}]
   [, <предложение 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
WAIT
ISOLATION LEVEL SNAPSHOT;
```

Транзакция с такими характеристиками режима доступа, режима разрешения блокировок, без задания средств резервирования таблиц и при уровне изоляции по умолчанию автоматически запускается, если пользователь, выполняя обращение к данным базы данных, не указал никакой транзакции.

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).

**Режим доступа**

При режиме доступа READ WRITE операции в контексте данной транзакции могут быть как операциями чтения, так и операциями изменения данных. Это режим транзакции по умолчанию. В режиме READ ONLY в контексте данной транзакции могут выполняться только операции выборки данных SELECT.

**Режим разрешения блокировок**

Есть два режима разрешения блокировок: WAIT и NO WAIT. В режиме WAIT (это режим по умолчанию) при появлении конфликта с параллельным процессом, выполняющим обновление данных в той же базе данных, такая транзакция будет ожидать завершения конкурирующей транзакции путем ее подтверждения (COMMIT) или отката (ROLLBACK). Этот режим дает отличные формы поведения в зависимости от уровня изоляции транзакций. Если для режима WAIT задать предложение LOCK TIMEOUT, то ожидание будет продолжаться только в течение указанного в этом предложении количества секунд. По истечении этого срока будет выдано сообщение об ошибке: "Lock time-out on wait transaction"(Истечение времени ожидания блокировки для транзакции WAIT).

Если установлен режим NO WAIT, то при появлении конфликта блокировки данная транзакция немедленно вызовет исключение базы данных.

**Уровень изоляции**

Уровень изоляции запускаемой транзакции задается необязательным предложением ISOLATION LEVEL. Эта самая важная характеристика транзакции, которая определяет ее поведение по отношению к другим одновременно выполняющимся транзакциям. Существует три уровня изоляции транзакции в СУБД Ред База Данных: SNAPSHOT, SNAPSHOT TABLE STABILITY и уровень изоляции READ COMMITTED с двумя уточнениями (NO RECORD_VERSION и RECORD_VERSION).

**Уровень изоляции** SNAPSHOT

Уровень изоляции SNAPSHOT (уровень изоляции по умолчанию) означает, что транзакции видны лишь те изменения базы данных, которые были выполнены и подтверждены другими одновременно выполняющимися транзакциями на момент старта данной транзакции. Любые подтвержденные изменения, сделанные другими конкурирующими транзакциями, не будут видны в такой транзакции. Чтобы увидеть эти изменения, нужно остановить транзакцию (подтвердить ее или выполнить полный откат) и запустить транзакцию заново.

**Уровень изоляции** SNAPSHOT TABLE STABILITY

Уровень изоляции транзакции SNAPSHOT TABLE STABILITY позволяет, как и в случае SNAPSHOT, видеть только те изменения, которые были выполнены в параллельных процессах на момент старта данной транзакции. При этом после старта транзакции в других клиентских транзакциях невозможно выполнение изменений в таблицах базы данных. Все такие попытки приведут к исключениям базы данных. Просматривать любые данные другие транзакции могут свободно. Только при помощи предложения резервирования (RESERVING) можно разрешить другим транзакциям изменять данные в указанных таблицах. Если на момент старта клиентом транзакции с уровнем изоляции SNAPSHOT TABLE STABILITY какая-нибудь другая транзакция выполнила неподтвержденное изменение данных любой таблицы базы данных, то запуск такой транзакции приведет к ошибке базы данных.

**Уровень изоляции** READ COMMITTED

Уровень изоляции READ COMMITTED позволяет в транзакции без ее перезапуска видеть все подтвержденные изменения данных базы данных, выполненные в параллельных процессах. Неподтвержденные изменения не видны в транзакции этого уровня изоляции. Для получения списка строк интересующей таблицы необходимо повторное выполнение оператора SELECT в рамках активной транзакции без перезапуска транзакции.

Для этого уровня изоляции существует две модификации характеристик.

В случае задания предложения NO RECORD_VERSION (значение по умолчанию) данная транзакция требует, чтобы были подтверждены все выполненные другими транзакциями измененные версии всех записей всех таблиц. Если при этом для транзакции задан режим разрешения блокировок WAIT, то такая транзакция просто ожидает подтверждения или отката всех других параллельных транзакций, выполнивших любые изменения данных в базе данных. При задании для варианта WAIT предложения LOCK TIMEOUT ожидание будет длиться не более указанного в предложении времени. Если же для транзакции установлен режим разрешения блокировок NO WAIT, то транзакция завер-

шается аварийно с выдачей исключения базы данных.

При задании RECORD_VERSION транзакция читает последнюю подтвержденную версию записей, независимо от того, существуют ли другие измененные и еще не подтвержденные версии записей. В этом случае режим разрешения блокировок (WAIT или NO WAIT) никак не влияет на поведение транзакции.

**Опция** NO AUTO UNDO

При задании этой опции в случае отката транзакции не ведётся журнал для отмены изменений. В таком случае сборка мусора выполнится в рамках других транзакций. Эта опция может оказаться полезной при выполнении операций массовых вставок, которые не должны откатываться. Если в транзакции не выполняется никаких изменений, опция NO AUTO UNDO игнорируется.

**Опция** IGNORE LIMBO

С предложением IGNORE LIMBO игнорируются записи, создаваемые Limbo транзакциями. Состояние Limbo транзакции означает, что она стартовала в режиме 'two phase commit' вызовом процедуры Prepare. Эта опция в основном используется утилитой gfix.

**Резервирование таблиц**

Предложение RESERVING резервирует указанные в списке таблицы, то есть запрещает другим транзакциям вносить в эти таблицы изменения или даже читать данные из этих таблиц в то время как выполняется данная транзакция. Либо, наоборот, в предложении можно указать список таблиц, в которые параллельные процессы могут вносить изменения.

```
RESERVING <предложение резервирования>

<предложение резервирования> ::=
   <имя таблицы> [, <имя таблицы> ...]
   [FOR [SHARED | PROTECTED] {READ | WRITE}]
   [, <предложение резервирования>] ...
```

Если опущено ключевое SHARED и PROTECTED, то предполагается SHARED. Если опущено предложение FOR, то предполагается FOR SHARED READ.

Допустимые варианты поведения параллельных транзакций в соответствии с заданными характеристиками текущей транзакции зависят и от уровня изоляции текущей транзакции.

При выполнении транзакции с уровнем изоляции SNAPSHOT для параллельных транзакций допустимы следующие варианты поведения:

- SHARED READ — не оказывает никакого влияния на выполнение параллельных транзакций;
- SHARED WRITE — на поведение параллельных транзакций с уровнями изоляции SNAPSHOT и READ COMMITTED не оказывает никакого влияния, для транзакций с уровнем изоляции SNAPSHOT TABLE STABILITY запрещает не только запись, но также и чтение данных из указанных таблиц;
- PROTECTED READ — допускает только чтение данных из резервируемых таблиц для параллельных транзакций с любым уровнем изоляции, попытка внесения изменений приводит к исключению базы данных;
- 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.

При выполнении транзакции с уровнем изоляции SNAPSHOT TABLE STABILITY для параллельных транзакций допустимы 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 транзакция имеет уровень доступа READ WRITE);
- SHARED WRITE — для параллельных транзакций с уровнем доступа READ WRITE и с уровнями изоляции SNAPSHOT и READ COMMITTED позволяет читать 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 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 и чтение данных из резервируемых таблиц.

При выполнении транзакции с уровнем изоляции READ CMMITTED для параллельных транзакций допустимы 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 (при уровне доступа READ WRITE);
- SHARED WRITE — для транзакций с уровнем доступа READ WRITE и с уровнями изоляции SNAPSHOT и READ COMMITTED позволяет читать 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 чтение данных из резервируемых таблиц для параллельных транзакций с любым уровнем изоляции;
- PROTECTED WRITE — для параллельных транзакций с уровнями изоляции SNAPSHOT и READ COMMITTED разрешает только чтение данных и запрещает запись в указанные в списке таблицы, для транзакций с уровнем изоляции SNAPSHOT TABLE STABILITY запрещает не только изменение данных, но и чтение данных из резервируемых таблиц.

Подробно оператор, работа с транзакциями, использование вложенных транзакций описаны в главе 10 «Транзакции».

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 Д.99. 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 Д.100. 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':]'
```

## Создание регулярных выражений

**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 Д.3.

Таблица Д.3 — 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) |

| Идентификатор | Описание |
|---|---|
| SPACE | Пробел (ASCII 32) |
| 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[^]+\(R[^]+\)' ESCAPE '\'   TRUE
'France[FR]' SIMILAR TO 'Fr[^]+#[F[^]+#]' ESCAPE '#'  TRUE
'Puerto-Rico' SIMILAR TO 'P%$-R%' ESCAPE '$'          TRUE
```

# UPDATE

Оператор `UPDATE` используется для изменения значений столбцов существующих строк таблицы или представления (таблиц, лежащих в основе представления). Синтаксис оператора:

**Листинг Д.101. Синтаксис оператора `UPDATE`**

```
UPDATE {<имя таблицы> | <имя представления>} [[AS] <псевдоним>]
SET <имя столбца> = <значение> [, <имя столбца> = <значение> ...]
[WHERE { <условие поиска> | CURRENT OF <имя курсора>}]
[PLAN <план>]
[ORDER BY <упорядочиваемый элемент> [, <упорядочиваемый элемент> ... ]]
[ROWS <значение 1> [TO <значение 2>]]
[RETURNING <имя столбца> [[AS] <алиас>] [, <имя столбца> [[AS] <алиас>]...]
   [INTO [:]<имя переменной> [, [:]<имя переменной> ...]] ];
```

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. Значением, как и в случае оператора `INSERT`, может быть сколь угодно сложное выражение. Значение определяется следующим синтаксисом:

```
<значение> ::= {
```

```
    <литерал>
  | <выражение>
  | <встроенная функция>
  | <UDF> [(<параметр> [, <параметр> ...])]
  | NEXT VALUE FOR <имя генератора>
  | (<выбор одного>) }
```

В SQL Ред База Данных существует два типа встроенных функций — обычные встроенные функции и агрегатные функции в операторе SELECT.

Обычная встроенная функция — это функция, получающая один или более параметров, которая не связана с оператором SQL выборки данных SELECT. Функция возвращает ровно одно значение. Параметры передаются таким функциям на основании принятого для каждой функции синтаксиса. Описание встроенных функций см. в Приложении Е.

Как определяется агрегатная функция в операторе SELECT можно посмотреть в листинге Д.76.

Конструкция NEXT VALUE FOR используется вместо функции GEN_ID.

Предложение WHERE определяет множество строк, к которым будет применяться операция изменения данных. Если это предложение не указано, будут изменены все существующие строки таблицы в том случае, если не указано также и предложение ORDER BY и предложение ROWS.

Подробнее об условиях поиска в предложении WHERE см. в разделе 8.1 «SELECT».

В операторе может быть использовано ключевое слово PLAN, задающее план выборки данных.

Предложение ORDER BY используется в том случае, если будет задано предложение ROWS. Предложение ORDER BY задает упорядочение результатов выборки. В нем указывается список столбцов, по которым происходит упорядочение, направление сортировки (по возрастанию или по убыванию) и порядок сортировки для строковых столбцов, если этот порядок отличается от принятого для данного столбца. После упорядочения выполняются изменения для указанных строк.

Синтаксис предложение ORDER BY:

```
ORDER BY <упорядочиваемый элемент> [, <упорядочиваемый элемент> ... ]

<упорядочиваемый элемент> ::=
   {<имя столбца>|<псевдоним столбца>|<номер столбца>|<произвольное выражение>}
   [COLLATE <порядок сортировки>]
   [ASC[ENDING] | DESC[ENDING]]
   [NULLS {FIRST | LAST}]
```

В предложении перечисляются столбцы, по которым нужно упорядочить строки набора данных перед выполнением изменений. Можно задавать только имена столбцов.

Ключевое слово ASCENDING задает упорядочение по возрастанию значений. Используется 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 задает порядок сортировки строкового столбца, если нужен порядок, отличный от того, который был установлен для этого столбца. Допустимые порядки сортировки для различных наборов символов — см. в приложении В «Наборы символов и порядок сортировки».

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.

Необязательное предложение ROWS задает диапазон строк, к которым будет применена операция изменения данных. Предложение ROWS можно использовать, только если задано и предложение ORDER BY.

```
ROWS <значение 1> [TO <значение 2>]
```

Значением здесь может быть число или выражение, возвращающее числовое значение. Число может

быть и дробным, в этом случае десятичные знаки просто отбрасываются без округления числа. Если используется выражение, то оно должно быть заключено в круглые скобки. Если в выражении присутствует и оператор SELECT, то он дополнительно должен быть заключен в круглые скобки.

Значение 1 задает количество включаемых в операцию обновления строк, упорядоченных в предложении ORDER BY, если не задан вариант TO. Это первые строки в упорядоченном списке.

Иначе значение 1 задает начальный номер строки в упорядоченном списке строк, если указан и вариант TO. Значение 2 в этом случае задает конечный номер выбираемой строки.

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 строк, указанных в предложении ROWS.

Необязательное предложение RETURNING указывает, что оператор возвращает значения заданных столбцов изменяемой строки. Если оператор изменяет более одной строки таблицы, то в этом случае возникнет ошибка базы данных. Ключевое слово INTO позволяет сохранить эти возвращенные значения во внутренних переменных триггера или хранимой процедуры.

Подробнее об операторе изменения данных см. в главе 8 «Операторы DML».

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 Д.102. Синтаксис оператора UPDATE OR INSERT**

```
UPDATE OR INSERT INTO
   {<имя таблицы> | <имя представления>} [(<имя столбца> [, <имя столбца> ...] )]
VALUES (<значение> [, <значение> ...] )
[MATCHING (<имя столбца> [, <имя столбца>] ...)
[RETURNING <имя столбца> [[AS] <алиас>] [, <имя столбца> [[AS] <алиас>]...]
   [INTO [:]<имя переменной> [, [:]<имя переменной> ...]] ];
```

Для выполнения оператора UPDATE OR INSERT пользователь должен иметь полномочия и UPDATE, и INSERT к таблице (представлению).

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.

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.

Если не задано ключевое слово MATCHING, то в списке столбцов 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 задан 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.

Предложение `RETURNING` возвращает значения указанных столбцов измененной или добавленной строки. Если происходит изменение более чем одной строки, то выдается сообщение об ошибке базы данных.

См. также операторы `INSERT`, `DELETE`, `UPDATE`, конструкцию `NEXT VALUE FOR`.

# Приложение E Функции

В этом разделе описаны встроенные функции SQL. Функции можно разделить на два типа — обычные встроенные функции и агрегатные функции. Обычные встроенные функции получают параметры и возвращают ровно одно значение. Параметром может быть литерал, имя столбца таблицы, предварительно определенный литерал. Агрегатные функции в операторе `SELECT` могут применяться в списке выбора оператора `SELECT`, а также в условии в предложении `WHERE`, в списке выбора оператора `SELECT`, где оператор `SELECT` должен быть заключен в круглые скобки. Эти функции выполняют действия с множеством значений столбцов таблицы из строк, полученных в результате выполнения запроса к таблице (таблицам).

## ABS()

Обычная математическая функция. Возвращает абсолютное значение числового параметра с плавающей точкой или параметра, который может быть преобразован в число с плавающей точкой.

**Листинг E.1. 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()

Обычная математическая функция. Возвращает арккосинус числового параметра с плавающей точкой или параметра, который может быть преобразован в число с плавающей точкой.

**Листинг E.2. 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 $\pi$. Если входной параметр имеет значение `NULL`, то возвращается также пустое значение.

## ACOSH()

Обычная математическая функция. Возвращает обратный гиперболический косинус числового параметра с плавающей точкой или параметра, который может быть преобразован в число с плавающей точкой.

**Листинг E.3. 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.

# ASCII_CHAR()

Обычная функция для работы со строками. Возвращает символ ASCII заданного числового параметра или параметра, который может быть преобразован в целое число. Если входным параметром задается дробное число, то происходит его правильное округление до целого числа. Результат возвращается в наборе символов NONE.

**Листинг Е.4. 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()

Обычная функция для работы со строками. Возвращает число, соответствующее коду ASCII первого (единственного) символа заданного строкового параметра или параметра, который может быть преобразован в строку.

**Листинг Е.5. 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().

# ASIN()

Обычная математическая функция. Возвращает арксинус числового параметра с плавающей точкой или параметра, который может быть преобразован в число с плавающей точкой.

**Листинг Е.6. 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 $\pi$. Если входной параметр имеет значение NULL, то возвращается также пустое значение.

# ASINH()

Обычная математическая функция. Возвращает обратный гиперболический синус числового параметра с плавающей точкой или параметра, который может быть преобразован в число с плавающей точкой.

**Листинг E.7. 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()

Обычная математическая функция. Возвращает арктангенс числового параметра с плавающей точкой или параметра, который может быть преобразован в число с плавающей точкой.

**Листинг E.8. 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 $\pi$ можно использовать функцию PI()) до числа $\pi/2$. Если входной параметр имеет значение NULL, то возвращается также пустое значение.

# ATAN2()

Обычная математическая функция. Возвращает арктангенс частного от деления первого числового параметра с плавающей точкой на второй числовой параметр.

**Листинг E.9. 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()

Обычная математическая функция. Возвращает обратный гиперболический тангенс числового параметра с плавающей точкой или параметра, который может быть преобразован в число с плавающей точкой.

**Листинг E.10. 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.

# AVG()

Агрегатная функция. Можно использовать в качестве оконной. Вычисляет среднее значение среди множества значений числового столбца или выражения. Синтаксис:

**Листинг Е.11. Синтаксис функции AVG**

```
AVG([ALL | DISTINCT] <значение>)
```

Функция используется в операторе SELECT, который выбирает из таблицы базы данных некоторое количество строк на основании условия WHERE.

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.

Значение может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются. Недопустимо использование выражения, возвращающего пустое значение NULL.

Если при выполнении оператора SELECT было получено нулевое количество записей, то функция возвращает пустое значение NULL.

См. также агрегатные функции MIN(), COUNT(), SUM(), LIST(), OVER().

# BIN_AND()

Обычная функция побитовых операций. Возвращает результат двоичной операции И над несколькими числовыми целочисленными параметрами.

**Листинг Е.12. Синтаксис функции BIN_AND**

```
BIN_AND(<значение 1> [, <значение 2>]...)
```

Если задан один входной параметр, то функция возвращает значение этого параметра. Выходной параметр — результат выполнения логической функции конъюнкции над несколькими целочисленными параметрами. Если один из входных параметров или все имеют значение NULL, то возвращается также пустое значение.

# BIN_NOT()

Обычная функция побитовых операций. Возвращает результат логического отрицания к каждому биту двоичного представления целочисленного параметра.

**Листинг Е.13. 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()

Обычная функция побитовых операций. Возвращает результат двоичной операции ИЛИ над несколькими числовыми целочисленными параметрами.

**Листинг E.14. Синтаксис функции BIN_OR**

```
BIN_OR(<значение 1> [, <значение 2>]...)
```

Если задан один входной параметр, то функция возвращает значение этого параметра. Выходной параметр — результат выполнения логической функции дизъюнкции над несколькими целочисленными параметрами. Если любой из входных параметров или все имеют значение NULL, то возвращается также пустое значение.

# BIN_SHL()

Обычная функция побитовых операций. Возвращает результат операции сдвига влево двоичных знаков целочисленного параметра.

**Листинг E.15. 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()

Обычная функция побитовых операций. Возвращает результат операции сдвига вправо двоичных знаков целочисленного параметра.

**Листинг E.16. 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()

Обычная функция побитовых операций. Возвращает результат двоичной операции исключающего ИЛИ над несколькими числовыми целочисленными параметрами.

**Листинг E.17. Синтаксис функции BIN_XOR**

```
BIN_XOR(<значение 1> [, <значение 2>]...)
```

Если задан один входной параметр, то функция возвращает значение этого параметра. Выходной параметр — результат выполнения логической функции исключающей дизъюнкции над несколькими целочисленными параметрами. Если любой из входных параметров или все имеют значение NULL, то возвращается также пустое значение.

# BIT_LENGTH()

Обычная функция для работы со строками. Возвращает количество битов, занимаемых входным параметром функции. Возвращаемым значением будет OCTET_LENGTH * 8. Функция возвращает количество битов в параметре любого типа данных.

**Листинг E.18. Синтаксис функции BIT_LENGTH**

```
BIT_LENGTH (<параметр функции>)
```

См. также функции CHARACTER_LENGTH(), OCTET_LENGTH().

# CASE-WHEN-ELSE()

Условное выражение. Дает возможность выбрать результирующее значение из множества различных выражений.

**Листинг E.19. 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 предло-

жении 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 E.20. 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.

# CAST()

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

**Листинг E.21. 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 2 «Типы данных Ред База Данных».

См. также функцию EXTRACT().

# CEIL|CEILING()

Обычная математическая функция. Возвращает наименьшее целое число, превышающее значение входного параметра.

**Листинг E.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().

# CHAR_TO_UUID()

Обычная 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 E.23. 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().

# CHARACTER_LENGTH()

Обычная функция для работы со строками. Функция CHARACTER_LENGTH возвращает количество символов в параметре любого типа данных.

**Листинг E.24. Синтаксис функции CHARACTER_LENGTH**

```
{CHARACTER_LENGTH | CHAR_LENGTH} (<параметр функции>)
```

См. также функции BIT_LENGTH(), OCTET_LENGTH().

# CHECK_DDL_RIGHTS()

Системная функция проверки DDL прав на объекты. Ее синтаксис:

**Листинг E.25. 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 E.26. 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().

# COALESCE()

Обычная условная функция. Возвращает первое по порядку непустое значение в списке.

**Листинг E.27. 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.

# CORR()

Агрегатная статистическая функция. К аргументу статистической функции не применимы параметры ALL и DISTINCT. Можно использовать в качестве оконной.

Функция CORR возвращает коэффициент корреляции для пары выражений, возвращающих числовые значения.

**Листинг E.28. Синтаксис функции CORR**

```
CORR(<выражение1>, <выражение2>)
```

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:

$$\frac{\text{COVAR_POP(<выражение1>, <выражение2>)}}{\text{STDDEV_POP(<выражение2>) * STDDEV_POP(<выражение1>)}}$$

В случае если выборка записей пустая или содержит только значения NULL, результат будет содержать NULL.

См. также функции OVER(), COVAR_POP(), COVAR_SAMP(), STDDEV_POP(), STDDEV_SAMP(), VAR_POP(), VAR_SAMP().

# COS()

Обычная математическая 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 E.29. Синтаксис функции COS**

```
COS(<параметр>)
```

# COSH()

Обычная математическая функция. Возвращает гиперболический косинус заданного параметра, указанного в радианах. Если входной параметр имеет значение NULL, то возвращается также пустое значение.

**Листинг E.30. Синтаксис функции COSH**

```
COSH(<параметр>)
```

# COT()

Обычная математическая функция. Возвращает значение 1, деленное на тангенс передаваемого функции значения. Другими словами - котангенс значения в радианах.

**Листинг E.31. Синтаксис функции COT**

```
COT(<числовой параметр>)
```

Если параметр имеет значение NULL, то возвращается также пустое значение.

# COUNT()

Агрегатная функция. Можно использовать в качестве оконной. Подсчитывает количество строк таблицы, которые удовлетворяют условию выборки данных. Синтаксис:

**Листинг E.32. Синтаксис функции COUNT**

```
COUNT ({* | [ALL | DISTINCT] <столбец>})
```

Функция возвращает результат типа BIGINT.

Функция используется в операторе SELECT, который выбирает из таблицы базы данных некоторое количество строк на основании условия WHERE.

Задание параметра * означает, что функция подсчитывает все полученные оператором SELECT строки.

Ключевое слово ALL указывает, что подсчитываются все значения заданного столбца, полученные оператором SELECT. Это ключевое слово предполагается по умолчанию. Такой вариант эквивалентен по результатам заданию параметра *.

Ключевое слово DISTINCT задает подсчет только отличающихся значений указанного столбца. Одинаковыми считаются и столбцы, содержащие пустое значение NULL.

См. также агрегатные функции MIN(), MAX(), AVG(), SUM(), LIST(), OVER().

# COVAR_POP()

Агрегатная статистическая функция. К аргументу статистической функции не применимы параметры ALL и DISTINCT. Можно использовать в качестве оконной.

Функция COVAR_POP возвращает ковариацию совокупности пар выражений с числовыми значениями.

**Листинг E.33. Синтаксис функции COVAR_POP**

```
COVAR_POP(<выражение1>, <выражение2>)
```

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:

$$\frac{\text{SUM(<выражение1> * <выражение2>)} - \frac{\text{SUM(<выражение1>) * SUM(<выражение2>)}}{\text{COUNT(*)}}}{\text{COUNT(*)}}$$

В случае если выборка записей пустая или содержит только значения NULL, результат будет содержать NULL.

См. также функции COUNT(), SUM(), OVER(), CORR(), COVAR_SAMP(), STDDEV_POP(), STDDEV_SAMP(), VAR_POP(), VAR_SAMP().

# COVAR_SAMP()

Агрегатная статистическая функция. К аргументу статистической функции не применимы параметры ALL и DISTINCT. Можно использовать в качестве оконной.

Функция COVAR_SAMP возвращает выборочную ковариацию пары выражений с числовыми значениями.

**Листинг E.34. Синтаксис функции COVAR_SAMP**

```
COVAR_SAMP(<выражение1>, <выражение2>)
```

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:

$$\frac{\text{SUM(<выражение1> * <выражение2>)} - \frac{\text{SUM(<выражение1>) * SUM(<выражение2>)}}{\text{COUNT(*)}}}{\text{COUNT(*)-1}}$$

В случае если выборка записей пустая, содержит только 1 запись или содержит только значения NULL, результат будет содержать NULL.

См. также функции COUNT(), SUM(), OVER(), CORR(), COVAR_POP(), STDDEV_POP(), STDDEV_SAMP(), VAR_POP(), VAR_SAMP().

# CPU_LOAD()

Возвращает среднюю загрузку процессора в течение заданного интервала. Синтаксис выглядит следующим образом:

**Листинг E.35. 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;
```

# CREATE_FILE()

Создает файл в директории, прописанной в directories.conf, и заполняет его BLOB данными.

**Листинг E.36. Синтаксис функции CREATE_FILE**

```
CREATE_FILE(<псевдоним директории>, <имя файла>, <BLOB - параметр>)
```

В directories.conf хранится реальный путь к директории с псевдонимом <псевдоним директории>. 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) 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().

# DAMLEV()

Обычная функция для работы со строками. Функция рассчитывает расстояние Дамерау — Левенштейна между двумя строками.

**Листинг E.37. 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
```

# DATEADD()

Обычная функция даты и времени. 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 Е.38. 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()

Обычная функция даты и времени. 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 Е.39. 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.

См. также функции DATEADD(), EXTRACT(), CAST().

# DECODE()

Обычная условная функция. Является сокращенным вариантом функции CASE-WHEN-ELSE. Дает возможность выбрать возвращаемое значение из множества различных выражений.

**Листинг E.40. 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.

# DELETE_FILE()

Удаляет файл, созданный функцией CREATE_FILE. Ее синтаксис:

**Листинг E.41. Синтаксис функции DELETE_FILE**

```
DELETE_FILE(<имя файла>)
```

Параметр <имя файла> — это строка в формате, создаваемом функцией CREATE_FILE. 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().

# EXP()

Обычная математическая 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 E.42. Синтаксис функции EXP**

```
EXP(<целое число>)
```

# EXTRACT()

Обычная функция даты и времени. Функция для типов данных даты (DATE), времени (TIME) и даты/времени (TIMESTAMP) позволяет выделять различные элементы даты и времени. Синтаксис функции:

**Листинг E.43. 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.

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.

См. также функции DATEADD(), DATEDIFF(), CAST().

# FLOOR()

Обычная математическая функция. FLOOR возвращает наибольшее целое число, меньшее или равное указанного числового выражения.

**Листинг Е.44. 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 CEILING().

# GEN_ID()

Обычная функция для работы с генераторами. Позволяет получить значение, хранящееся в генераторе. Синтаксис функции:

**Листинг Е.45. 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>
```

# GEN_UUID()

Обычная функция для работы с UUID. Позволяет получить уникальное символьное значение в текущей базе данных, не повторяющееся ни при каких обращениях к этой функции. Синтаксис функции:

**Листинг E.46. Синтаксис функции GEN_UUID**

```
GEN_UUID()
```

Возвращаемое значение содержит 16 символов.

См. также функции CHAR_TO_UUID(), UUID_TO_CHAR().

# HASH()

Обычная функция для работы со строками. Функция возвращает хэш-значение, соответствующее входной строке, используя для этого встроенный алгоритм хэширования. Синтаксис:

**Листинг E.47. Синтаксис функции HASH**

```
HASH(<входной параметр>)
```

Функция поддерживает тип данных BLOB.

См. также функцию HASH_CP().

# 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 E.48. 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.

Пример:

```
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
```

# HASH_CP()

Обычная функция для работы со строками. 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 Е.49. 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 Е.50. 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(), OVER().

# IIF()

Обычная условная функция. Проверяет условие, если оно истинно, то возвращает первое значение, иначе — второе.

**Листинг Е.51. Синтаксис функции IIF**

```
IIF (<условие>, <значение 1> <значение 2>)
```

См. также функции CASE-WHEN-ELSE(), NULLIF(), COALESCE(), операторы IF-THEN-ELSE, WHILE-DO.

# LDAP_ATTR()

Читает указанный аттрибут из LDAP записи текущего пользователя.

**Листинг E.52. 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;
```

# LEFT()

Обычная функция для работы со строками. Возвращает указанные первые символы строки. Синтаксис:

**Листинг E.53. 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 правильное 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()`.

# LIST()

Агрегатная функция. Можно использовать в качестве оконной. Объединяет в один объект типа `BLOB` все данные, полученные из указанного выражения. Синтаксис функции:

**Листинг E.54. Синтаксис функции LIST**

```
LIST ([ALL | DISTINCT] <выражение> [, '<разделитель>'])
```

Ключевое слово `ALL` (значение по умолчанию) указывает, что в список попадают все значения, полученные оператором `SELECT`. Ключевое слово `DISTINCT` позволяет включить в список только отличающиеся значения.

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.

См. также агрегатные функции `MIN()`, `MAX()`, `AVG()`, `COUNT()`, `SUM()`, `OVER()`.

# LN()

Обычная математическая функция. Возвращает натуральный логарифм числа. Синтаксис:

**Листинг E.55. Синтаксис функции LN**

```
LN(<числовой параметр>)
```

Входной параметр — положительное число типа данных DOUBLE PRECISION. Выходное значение также имеет тип данных DOUBLE PRECISION.

# LOG()

Обычная математическая функция. Возвращает логарифм числа по заданному основанию. Синтаксис:

**Листинг E.56. 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()

Обычная математическая функция. Возвращает десятичный логарифм числового параметра. Синтаксис:

**Листинг E.57. 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().

# LOWER()

Обычная функция для работы со строками. Переводит все буквы строки в нижний регистр. Синтаксис функции:

**Листинг E.58. Синтаксис функции LOWER**

```
LOWER (<строка>)
```

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.

См. также строковые функции UPPER(), TRIM().

# LPAD()

Обычная функция для работы со строками. Возвращает строку определенного вида. Синтаксис:

**Листинг Е.59. 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>).

```
LPAD ('Hello' , 12)                 '       Hello'
LPAD ('World' , 12, ',')            ',,,,,,World'
LPAD ('Hello' , 12, '')             'Hello'
LPAD ('World' , 12, 'abc')          'abcabcaWorld'
LPAD ('Hello' , 12, 'abcdefghij')   'abcdefgHello'
LPAD ('World' , 2)                  'Wo'
LPAD ('Hello' , 2, ',')             'He'
LPAD ('World' , 2, '')              'Wo'
```

См. также строковую функцию RPAD().

# MAX()

Агрегатная функция. Можно использовать в качестве оконной. Отыскивает максимальное значение столбца в таблице. Применима к любому типу данных. Синтаксис функции:

**Листинг Е.60. Синтаксис функции MAX**

```
MAX ([ALL | DISTINCT] <значение>)
```

Функция используется в операторе SELECT, который выбирает из таблицы базы данных некоторое количество строк на основании условия WHERE.

Ключевое слово ALL указывает, что в операции поиска максимального значения принимают все значения заданного столбца, полученные оператором SELECT. Это ключевое слово предполагается по умолчанию.

Ключевое слово DISTINCT задает предварительное удаление из списка для поиска максимального значения всех дублированных значений.

Если при выполнении оператора SELECT было получено нулевое количество записей, то функция возвращает пустое значение NULL.

См. также агрегатные функции MIN(), MINVALUE(), MAXVALUE(), OVER().

# MAXVALUE()

Обычная условная функция. Отыскивает максимальное значение в заданном списке. Применима к любому типу данных. Синтаксис функции:

**Листинг E.61. Синтаксис функции MAXVALUE**

```
MAXVALUE (<значение> [, <значение>]...)
```

См. также функции MIN(), MAX(), MINVALUE().

# MIN()

Агрегатная функция. Можно использовать в качестве оконной. Отыскивает минимальное значение столбца в таблице. Применима к любому типу данных. Синтаксис функции:

**Листинг E.62. Синтаксис функции MIN**

```
MIN ([ALL | DISTINCT] <значение>)
```

Функция используется в операторе SELECT, который выбирает из таблицы базы данных некоторое количество строк на основании условия WHERE.

Ключевое слово ALL указывает, что в операции поиска минимального значения принимают все значения заданного столбца, полученные оператором SELECT. Это ключевое слово предполагается по умолчанию.

Ключевое слово DISTINCT задает предварительное удаление из списка для поиска минимального значения всех дублированных значений.

Если при выполнении оператора SELECT было получено нулевое количество записей, то функция возвращает пустое значение NULL.

См. также агрегатные функции MAX(), MINVALUE(), MAXVALUE(), OVER().

# MINVALUE()

Обычная функция. Отыскивает минимальное значение в заданном списке. Применима к любому типу данных. Синтаксис функции:

**Листинг E.63. Синтаксис функции MINVALUE**

```
MINVALUE (<значение> [, <значение>]...)
```

См. также функции MIN(), MAX(), MAXVALUE().

# MOD()

Обычная математическая функция. Возвращает остаток от деления первого целочисленного параметра на второй целочисленный параметр. Синтаксис:

**Листинг E.64. Синтаксис функции MOD**

```
MOD(<числовой параметр 1>, <числовой параметр 2>)
```

# NEXT VALUE FOR

Конструкция (функция) NEXT VALUE FOR позволяет получить значение указанного генератора, увеличенное на единицу. Синтаксис:

**Листинг E.65. 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 6 «Работа с генераторами».

См. также операторы CREATE GENERATOR, CREATE SEQUENCE, DROP GENERATOR, DROP SEQUENCE, SET GENERATOR, ALTER SEQUENCE, функцию GEN_ID().

# NULLIF()

Обычная условная функция. Проверяет два значения. Если они равны, возвращает NULL, иначе первое значение.

**Листинг E.66. Синтаксис функции NULLIF**

```
NULLIF(<значение 1>, <значение 2>)
```

См. также функции CASE-WHEN-ELSE(), IIF(), COALESCE(), операторы IF-THEN-ELSE, WHILE-DO.

# OCTET_LENGTH()

Обычная функция для работы со строками. Возвращает количество байтов, занимаемых входным параметром функции. Функция возвращает количество байтов в параметре любого типа данных.

**Листинг E.67. Синтаксис функции OCTET_LENGTH**

```
OCTET_LENGTH(<параметр функции>)
```

См. также функции BIT_LENGTH(), CHARACTER_LENGTH().

# OVER()

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.

**Листинг E.68. Синтаксис оконных функций**

```
<оконная функция> ::= <имя оконной функции>([ <выражение> [, <выражение> ...]])
```

```
                        OVER ([<выражение секционирования>] [<выражение сортировки>])

<выражение секционирования> ::= PARTITION BY <выражение> [, <выражение> ...]

<выражение сортировки> ::= ORDER BY <выражение> [{ASC|DESC}] [NULLS {FIRST|LAST}]
                           [,<выражение> [{ASC|DESC}] [NULLS {FIRST|LAST}]...]

<имя оконной функции>::=
    <агрегатная функция>
  | <ранжирующая функция>
  | <навигационная функция>

<ранжирующая функция> ::= RANK | DENSE_RANK | ROW_NUMBER

<навигационная функция> ::= LEAD | LAG | FIRST_VALUE | LAST_VALUE | NTH_VALUE
```

Выражение может содержать столбец таблицы, константу, переменную, выражение, скалярную или агрегатную функцию. Оконные функции в качестве выражения не допускаются.

## Агрегатные функции

Все агрегатные функции могут быть использованы в качестве оконных функций, при добавлении предложения `OVER`.

## Секционирование

Как и для агрегатных функций, которые могут работать отдельно или по отношению к группе, оконные функции тоже могут работать для групп, которые называются "секциями"(partition) или разделами.

Для каждой строки, оконная функция обсчитывает только строки, которые попадают в то 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.

## Сортировка

Предложение `ORDER BY` может быть использовано с секционированием или без него. Предложение `ORDER BY` внутри `OVER` задаёт порядок, в котором оконная функция будет обрабатывать строки. Этот порядок не обязан совпадать с порядком вывода строк.

Есть ещё одно важное понятие, связанное с оконными функциями: для каждой строки существует набор строк в её разделе, называемый рамкой окна (кадры окна). По умолчанию, с указанием `ORDER BY` рамка состоит из всех строк от начала раздела до текущей строки и строк, равных текущей по значению выражения `ORDER BY`. Без `ORDER BY` рамка по умолчанию состоит из всех строк раздела.

Таким образом, для стандартных агрегатных функций, предложение `ORDER BY` заставляет возвращать частичные результаты агрегации по мере обработки записей.

Вы можете использовать несколько окон с различными сортировками, и дополнять предложение `ORDER BY` опциями `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 предложение `ORDER BY`, за исключением `LIST()`.

## 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.

Функция DENSE_RANK 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.

Функция RANK 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.

Функция ROW_NUMBER возвращает последовательный номер строки в секции результирующего набора, где 1 соответствует первой строке в каждой из секций.

## Навигационные функции

Навигационные функции получают простые (не агрегированные) значения выражения из другой строки запроса в той же секции.

Функция FIRST_VALUE(<выражение>) возвращает первое значение из упорядоченного набора значений. Параметр <выражение> может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF.

Функция LAG(<выражение> [,<смещение> [, <default>]]) 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. Параметр <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.

Функция LAST_VALUE(<выражение>) возвращает последнее значение из упорядоченного набора значений. Параметр <выражение> может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF.

Функция LEAD(<выражение> [,<смещение> [, <default>]]) 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. Параметр <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.

Функция NTH_VALUE(<выражение> [,<смещение>]) [FROM FIRST | FROM LAST] 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 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.

Подробнее об оконных функциях см. в главе 8.

# OVERLAY()

Обычная строковая функция. Синтаксис:

**Листинг E.69. 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> + <количество заменяемых
символов>)
```

См. также строковые функции UPPER(), LOWER(), TRIM(), LPAD(), SUBSTRING(), REPLACE().

# PI()

Обычная математическая функция. Возвращает число «пи» (3.1459...) типа данных DOUBLE PRECISION.

# POSITION()

Обычная строковая функция. Отыскивает позицию подстроки в исходной строке. Существует два варианта синтаксиса:

**Листинг E.70. Синтаксис функции POSITION**

```
POSITION(<строка 1> IN <строка 2>)
POSITION(<строка 1>, <строка 2> [,<начальная позиция>])
```

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 игнорирую любые вхождения подстроки в исходную строку до этой позиции.

# POWER()

Обычная математическая функция. Производит возведение числа в степень. Синтаксис:

**Листинг E.71. 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()

Обычная математическая функция. Возвращает случайное число типа данных DOUBLE PRECISION в диапазоне от 0 до 1.

# 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 E.72. 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 E.1.

Таблица E.1 — Контекстные переменные в пространстве имён SYSTEM

| Имя переменной | Описание |
|---|---|
| ENGINE_VERSION | Версия сервера (например, 3.0.11) |
| FULL_VERSION | Полная версия сборки СУБД (например, WI-V3.0.11.0 RedDatabase 3.0 SNAPSHOT.16 (9ec7320661241a96270a45741e9aae609d024ade)) |
| EDITION | Установленная редакция СУБД Ред База Данных: Open, Standard или Enterprise |

| Имя переменной | Описание |
|---|---|
| DB_NAME | Полный путь к базе данных или, если подключение через путь запрещено, алиас |
| REPLICA | Является ли текущая база репликой (true) или нет |
| REPLICATION_SEQUENCE | Текущее значение последовательности репликации (номер последнего сегмента, записанного в журнал репликации) |
| 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. |
| CLIENT_ADDRESS | Для TCPv4 — IP адрес, для XNET — локальный ID процесса. Для всех остальных протоколов переменная имеет значение NULL |
| CLIENT_HOST | Имя хоста сетевого протокола 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) |
| CURRENT_USER | Глобальная переменная CURRENT_USER |
| CURRENT_ROLE | Глобальная переменная CURRENT_ROLE |
| 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. |
| 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 |
| 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 предопределенных переменных, показанных в таблице E.2.

Таблица E.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 E.3.

Таблица E.3 — Контекстные переменные в пространстве имён AUTHDATA

| Имя переменной | Описание |
|---|---|
| AUTH_TYPE | Тип аутентификации: SECURITY или LDAP |
| AUTH_PLUGIN | Плагин аутентификации:<br>• Legacy_Auth;<br>• Srp;<br>• Multifactor;<br>• Win_Sspi;<br>• Gss. |
| 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 E.73. 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().

# READ_FILE()

Функция читает файл из директории и возвращает данные типа BLOB. Ее синтаксис:

**Листинг E.74. Синтаксис функции READ_FILE**

```
READ_FILE(<имя файла>)
```

Параметр <имя файла> — это строка в формате, создаваемом функцией CREATE_FILE.

```
select READ_FILE('test_dir/20150708/text-UnceByjB8Nba1Bbo6+h9lS.txt') from
rdb$database;
```

См. также функции CREATE_FILE(),DELETE_FILE().

# REGEXP_SUBSTR()

Строковая функция REGEXP_SUBSTR расширяет функциональность SUBSTRING(), позволяя искать подстроку, которая соответствует регулярному выражению. Синтаксис функции выглядит следующим образом:

**Листинг E.75. Синтаксис функции REGEXP_SUBSTR**

```
REGEXP_SUBSTR(<исходная строка>, <шаблон>[,<номер группы>, <номер вхождения>,
<параметр сравнения>])
```

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. Для проверки соответствия шаблона регулярному выражения см. синтаксис предиката SIMILAR TO. В качестве символа экранирования используется обратный слэш '\'.
- <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.
- <номер вхождения> — это число показывает сколько нужно искать вхождений подстроки в исходной строке. 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 – начало 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 | Замечание |
|---|---|---|
| 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. |
| S | Режим единственной строки (single-line). | При этом в шаблоне символы _ и % начинают соответствовать символам возврата каретки и переноса строки. |
| 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. |

| Параметр | Описание | Замечание |
|---|---|---|
| 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. |

Примеры использования функции:

```
REGEXP_SUBSTR('abcd1234efgh5678', '([[:ALPHA:]]+[0-9]+)',1,1)      abcd1234
REGEXP_SUBSTR('abcd1234efgh5678', '([[:ALPHA:]]+[0-9]+)',1,5)      efgh5678
REGEXP_SUBSTR('abcd1234efgh5678', '([[:ALPHA:]]+[0-9]+)',1,5,'G') efgh5
```

# REGR_AVGX()

Агрегатная функция линейной регрессии. Можно использовать в качестве оконной.
Функция REGR_AVGX вычисляет среднее независимой переменной линии регрессии.

**Листинг E.76. Синтаксис функции REGR_AVGX**

```
REGR_AVGX(y, x)
```

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:

```
    SUM(<X>)
-------------------
REGR_COUNT(y, x)
<X> ::= CASE WHEN x IS NOT NULL AND y IS NOT NULL THEN x END
```

См. также функции SUM(), OVER(), REGR_AVGY(), REGR_COUNT(), REGR_INTERCEPT(), REGR_R2(), REGR_SLOPE(), REGR_SXX(), REGR_SXY(), REGR_SYY().

# REGR_AVGY()

Агрегатная функция линейной регрессии. Можно использовать в качестве оконной.
Функция REGR_AVGY вычисляет среднее зависимой переменной линии регрессии.

**Листинг E.77. Синтаксис функции REGR_AVGY**

```
REGR_AVGY(y, x)
```

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:

```
    SUM(<Y>)
-------------------
REGR_COUNT(y, x)
<Y> ::= CASE WHEN x IS NOT NULL AND y IS NOT NULL THEN y END
```

См. также функции SUM(), OVER(), REGR_AVGX(), REGR_COUNT(), REGR_INTERCEPT(), REGR_R2(), REGR_SLOPE(), REGR_SXX(), REGR_SXY(), REGR_SYY().

# REGR_COUNT()

Агрегатная функция линейной регрессии. Можно использовать в качестве оконной.

Функция REGR_COUNT возвращает количество непустых пар, используемых для создания линии регрессии.

**Листинг E.78. Синтаксис функции REGR_COUNT**

```
REGR_COUNT(y, x)
```

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:

```
SUM(CASE WHEN x IS NOT NULL AND y IS NOT NULL THEN 1 END)
```

См. также функции SUM(), OVER(), REGR_AVGX(), REGR_AVGY(), REGR_INTERCEPT(), REGR_R2(), REGR_SLOPE(), REGR_SXX(), REGR_SXY(), REGR_SYY().

# REGR_INTERCEPT()

Агрегатная функция линейной регрессии. Можно использовать в качестве оконной.

Функция REGR_INTERCEPT вычисляет точку пересечения линии регрессии с осью Y.

**Листинг E.79. Синтаксис функции REGR_INTERCEPT**

```
REGR_INTERCEPT(y, x)
```

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_AVGY(y, x) - REGR_SLOPE(y, x) * REGR_AVGX(y, x)
```

См. также функции OVER(), REGR_AVGX(), REGR_AVGY(), REGR_COUNT(), REGR_R2(), REGR_SLOPE(), REGR_SXX(), REGR_SXY(), REGR_SYY().

# REGR_R2()

Агрегатная функция линейной регрессии. Можно использовать в качестве оконной.

Функция REGR_R2 вычисляет коэффициент детерминации, или R-квадрат, линии регрессии.

**Листинг E.80. Синтаксис функции REGR_R2**

```
REGR_R2(y, x)
```

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:

```
POWER(CORR(y, x), 2)
```

См. также функции CORR(), OVER(), REGR_AVGX(), REGR_AVGY(), REGR_COUNT(), REGR_INTERCEPT(), REGR_SLOPE(), REGR_SXX(), REGR_SXY(), REGR_SYY().

# REGR_SLOPE()

Агрегатная функция линейной регрессии. Можно использовать в качестве оконной.

Функция REGR_SLOPE вычисляет угол наклона линии регрессии.

**Листинг E.81. Синтаксис функции REGR_SLOPE**

```
REGR_SLOPE(y, x)
```

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:

```
COVAR_POP(y, x)
  VAR_POP(<X>)
<X> ::= CASE WHEN x IS NOT NULL AND y IS NOT NULL THEN x END
```

См. также функции COVAR_POP(), VAR_POP(), OVER(), REGR_AVGX(), REGR_AVGY(), REGR_COUNT(), REGR_INTERCEPT(), REGR_R2(), REGR_SXX(), REGR_SXY(), REGR_SYY().

# REGR_SXX()

Агрегатная функция линейной регрессии. Можно использовать в качестве оконной.

Функция REGR_SXX показывает диагностическую статистику, используемую для анализа регрессии.

**Листинг E.82. Синтаксис функции REGR_SXX**

```
REGR_SXX(y, x)
```

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(<X>)
<X> ::= CASE WHEN x IS NOT NULL AND y IS NOT NULL THEN x END
```

См. также функции VAR_POP(), OVER(), REGR_AVGX(), REGR_AVGY(), REGR_COUNT(), REGR_INTERCEPT(), REGR_R2(), REGR_SLOPE(), REGR_SXY(), REGR_SYY().

# REGR_SXY()

Агрегатная функция линейной регрессии. Можно использовать в качестве оконной.

Функция REGR_SXY показывает диагностическую статистику, используемую для анализа регрессии.

**Листинг E.83. Синтаксис функции REGR_SXY**

```
REGR_SXY(y, x)
```

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(),OVER(),REGR_AVGX(), REGR_AVGY(), REGR_COUNT(), REGR_INTERCEPT(), REGR_R2(), REGR_SLOPE(), REGR_SXX(), REGR_SYY().

# REGR_SYY()

Агрегатная функция линейной регрессии. Можно использовать в качестве оконной.

Функция REGR_SYY показывает диагностическую статистику, используемую для анализа регрессии.

**Листинг E.84. Синтаксис функции REGR_SYY**

```
REGR_SYY(y, x)
```

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(),OVER(),REGR_AVGX(), REGR_AVGY(), REGR_COUNT(), REGR_INTERCEPT(), REGR_R2(), REGR_SLOPE(), REGR_SXX(), REGR_SXY().

# REPLACE()

Обычная строковая функция. Отыскивает подстроку в исходной строке и заменяет на другую. Синтаксис:

**Листинг Е.85. 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()

Обычная строковая функция. Синтаксис:

**Листинг Е.86. Синтаксис функции REVERSE**

```
REVERSE(<строка>)
```

Функция возвращает значение строкового параметра, где символы располагаются в обратном порядке.

# RIGHT()

Обычная строковая функция. Возвращает указанные последние символы строки. Синтаксис:

**Листинг Е.87. 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 правильное 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().

# ROUND()

Обычная математическая функция. Выполняет округление числа. Синтаксис:

**Листинг E.88. 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.

# RPAD()

Обычная строковая функция. Возвращает строку определенного вида. Синтаксис:

**Листинг E.89. 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 права 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>).

```
RPAD ('Hello' , 12)                 |  '       Hello'
RPAD ('World' , 12, ',')            |  'World,,,,,,'
RPAD ('Hello' , 12, '')             |  'Hello'
RPAD ('World' , 12, 'abc')          |  'Worldabcabca'
RPAD ('Hello' , 12, 'abcdefghij')   |  'Helloabcdefg'
RPAD ('World' , 2)                  |  'Wo'
RPAD ('Hello' , 2, ',')             |  'He'
RPAD ('World' , 2, '')              |  'Wo'
```

См. также строковую функцию LPAD().

# SIGN()

Обычная математическая функция. Синтаксис:

**Листинг E.90. Синтаксис функции SIGN**

```
SIGN(<числовой параметр>)
```

Функция возвращает +1, если число положительное, 0, если число равно нулю, и ˇ1, если число отрицательное.

# SIN()

Обычная математическая функция. Возвращает синус заданного параметра в радианах. Если входной параметр имеет значение NULL, то возвращается также пустое значение.

**Листинг E.91. Синтаксис функции SIN**

```
SIN(<числовой параметр>)
```

# SINH()

Обычная математическая функция. Возвращает гиперболический синус заданного параметра в радианах. Если входной параметр имеет значение NULL, то возвращается также пустое значение.

**Листинг E.92. Синтаксис функции SINH**

```
SINH(<числовой параметр>)
```

# SQRT()

Обычная математическая функция. Возвращает квадратный корень заданного параметра. Если входной параметр имеет значение NULL, то возвращается также пустое значение.

**Листинг E.93. Синтаксис функции SQRT**

```
SQRT(<числовой параметр>)
```

Входной параметр и возвращаемое значение имею тип данных DOUBLE PRECISION.

# STDDEV_POP()

Агрегатная статистическая функция. К аргументу статистической функции не применимы параметры ALL и DISTINCT. Можно использовать в качестве оконной.

Функция STDDEV_POP возвращает среднеквадратичное отклонение для группы. Значения NULL пропускаются.

**Листинг E.94. Синтаксис функции STDDEV_POP**

```
STDDEV_POP(<выражение>)
```

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 OVER(), COVAR_POP(), COVAR_SAMP(), CORR(), STDDEV_SAMP(), VAR_POP(), VAR_SAMP().

# STDDEV_SAMP()

Агрегатная статистическая функция. К аргументу статистической функции не применимы параметры ALL и DISTINCT. Можно использовать в качестве оконной.

Функция STDDEV_SAMP возвращает стандартное отклонение для группы. Значения NULL пропускаются.

**Листинг E.95. Синтаксис функции STDDEV_SAMP**

```
STDDEV_SAMP(<выражение>)
```

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 OVER(), COVAR_POP(), COVAR_SAMP(), CORR(), STDDEV_POP(), VAR_POP(), VAR_SAMP().

# SUBSTRING()

Обычная строковая функция. Выделяет подстроку. Синтаксис функции:

**Листинг E.96. Синтаксис функции 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>
```

Функция SUBSTRING возвращает подстроку строки начиная с указанной позиции (FROM, позиции в исходной строке нумеруются, начиная с единицы) с заданным количеством символов (FOR). 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.

| | |
|---|---|
| SUBSTRING ('Руководство ' FROM 5 FOR 3) | вод |
| SUBSTRING('abcdefg' SIMILAR 'a#"bcde#"fg' ESCAPE '#' ) | bcde |
| SUBSTRING('abcdefg' SIMILAR 'a#"%#"' ESCAPE '#' ) | bcdefg |
| SUBSTRING('abcdefg' SIMILAR '_#"%#"_' ESCAPE '#' ) | bcdef |
| SUBSTRING('abcdefg' SIMILAR '#"abc#"%' ESCAPE '#' ) | abc |
| SUBSTRING('abcdefg' SIMILAR '#"abc#"' ESCAPE '#' ) | <null> |

См. также строковые функции UPPER(), LOWER(), TRIM(), LPAD(), OVERLAY().

# SUM()

Агрегатная функция. Можно использовать в качестве оконной. Функция возвращает сумму указанных значений полученных строк. Здесь значением может быть имя столбца, константа, переменная, выражение, неагрегатная функция или UDF, имеющие числовой тип данных.

**Листинг E.97. Синтаксис функции SUM**

```
SUM ({[ALL] <значение> | DISTINCT <значение>})
```

Функция используется в операторе SELECT, который выбирает из таблицы базы данных некоторое количество строк на основании условия WHERE.

Необязательное ключевое слово ALL указывает, что суммируются все значения, полученные оператором SELECT. Этот вариант предполагается по умолчанию.

Ключевое слово DISTINCT задает суммирование только отличающихся значений. При этом одинаковыми считаются и значения NULL.

Если при выполнении оператора SELECT было получено нулевое количество записей, то функция возвращает пустое значение NULL.

См. также агрегатные функции MIN(), MAX(), AVG(), COUNT(), LIST(), OVER().

# TAN()

Обычная математическая функция. Возвращает тангенс заданного параметра в радианах. Если входной параметр имеет значение NULL, то возвращается также пустое значение.

**Листинг E.98. Синтаксис функции TAN**

```
TAN(<числовой параметр>)
```

# TANH()

Обычная математическая функция. Возвращает гиперболический тангенс заданного параметра. Если входной параметр имеет значение NULL, то возвращается также пустое значение.

**Листинг E.99. Синтаксис функции TANH**

```
TANH(<числовой параметр>)
```

# TRIM()

Обычная функция для работы со строками. Удаляет начальные и/или конечные указанные символы (обычно пробелы) в исходной строке.

**Листинг E.100. Синтаксис функции TRIM**

```
TRIM ([[<спецификация удаления>] [<удаляемые символы>] FROM] <строка>)
<спецификация удаления> ::= LEADING | TRAILING | BOTH
```

Спецификация удаления определяет, из какой части строки (начальной или конечной) будут удаляться указанные символы:

- `LEADING` — символы удаляются из начальной части строки;
- `TRAILING` — удаляются конечные символы строки;
- `BOTH` (значение по умолчанию) — символы удаляются из начальной и из конечной части стоки.

Удаляемые символы — строка, содержащая произвольное количество символов (не обязательно только один), но ее длина не должна быть больше 4 Гбайт. Строка, как обычно, заключается в апострофы.

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.

См. также функции `BIT_LENGTH()`, `CHARACTER_LENGTH()`, `OCTET_LENGTH()`.

# TRUNC()

Обычная математическая 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 E.101. 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).

# UPPER()

Обычная строковая функция. Переводит все буквы строки в верхний регистр. Синтаксис функции:

**Листинг E.102. Синтаксис функции UPPER**

```
UPPER(<строка>)
```

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.

См. также строковые функции LOWER(), TRIM().

# UTC_TIMESTAMP()

Возвращает текущую дату и время по стандарту UTC в качестве значения в формате YYYY-MM-DD HH:MM:SS. Синтаксис:

**Листинг E.103. Синтаксис функции UTC_TIMESTAMP**

```
UTC_TIMESTAMP()
```

# UUID_TO_CHAR()

Обычная 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(32).

**Листинг E.104. 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().

# VAR_POP()

Агрегатная статистическая функция. К аргументу статистической функции не применимы параметры ALL и DISTINCT. Можно использовать в качестве оконной.

Функция VAR_POP возвращает выборочную дисперсию для группы. Значения NULL пропускаются.

**Листинг E.105. Синтаксис функции VAR_POP**

```
VAR_POP(<выражение>)
```

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:

$$\frac{\texttt{SUM(<выражение> * <выражение>)} - \frac{\texttt{SUM(<выражение>) * SUM(<выражение>)}}{\texttt{COUNT(<выражение>)}}}{\texttt{COUNT(<выражение>)}}$$

В случае если выборка записей пустая или содержит только значения NULL, результат будет содержать NULL.

См. также функции COUNT(), SUM(), OVER(), COVAR_POP(), COVAR_SAMP(), CORR(), STDDEV_SAMP(), STDDEV_POP(), VAR_SAMP().

# VAR_SAMP()

Агрегатная статистическая функция. К аргументу статистической функции не применимы параметры ALL и DISTINCT. Можно использовать в качестве оконной.

Функция VAR_SAMP возвращает несмещённую выборочную дисперсию для группы. Значения NULL пропускаются.

**Листинг Е.106. Синтаксис функции VAR_SAMP**

```
VAR_SAMP(<выражение>)
```

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:

$$\frac{\texttt{SUM(<выражение>) * SUM(<выражение>)} - \frac{\texttt{SUM(<выражение>) * SUM(<выражение>)}}{\texttt{COUNT(<выражение>)}}}{\texttt{COUNT(<выражение>)-1}}$$

В случае если выборка записей пустая, содержит только 1 запись или содержит только значения NULL, результат будет содержать NULL.

См. также функции COUNT(), SUM(), OVER(), COVAR_POP(), COVAR_SAMP(), CORR(), STDDEV_SAMP(), STDDEV_POP(), VAR_POP().

# Приложение Ж Контекстные переменные

В Ред База Данных существуют контекстные переменные, которые возвращают некоторые значения.

## 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(31)` 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` 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` 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(31)` возвращает имя пользователя, который в настоящий момент соединен с базой данных. Возвращаемое значение точно такое же, как и при использовании контекстной переменной `USER`.

# 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.

# 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$database;
-----------------------------------------
2015-07-03
select CAST('now' as TIMESTAMP) from rdb$database;
-------------------------------------------------
2015-07-03 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

ROW_COUNT типа INTEGER указывает общее количество строк, которые были прочитаны, добавлены, изменены или удалены в процессе выполнения предыдущего оператора SQL. Эта контекстная переменная может быть использована только в триггерах, хранимых процедурах и исполняемых блоках.

# 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!'
              ELSE 'Ошибок нет!';
           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(31). 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 З Операторы языка хранимых процедур, функций и триггеров

## CLOSE

Закрывает набор данных, связанный с данным курсором.

**Листинг З.1. 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 З.2. 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 З.3).

**Листинг З.3. 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 представлен в листинге 3.4.

**Листинг 3.4. 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. Кроме того, подпрограмма не может вызывать себя рекурсивно. Это может быть разрешено в будущем.

См. также операторы DECLARE VARIABLE, DECLARE PROCEDURE.

# DECLARE PROCEDURE

В хранимых функциях и хранимых процедурах можно объявлять подпроцедуры. Синтаксис оператора представлен в листинге 3.5.

**Листинг 3.5. 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. Кроме того, подпрограмма не может вызывать себя рекурсивно. Это может быть разрешено в будущем.

См. также операторы DECLARE VARIABLE, DECLARE FUNCTION.

# DECLARE VARIABLE

Оператор позволяет определить локальную переменную или курсор.

**Листинг 3.6. Синтаксис оператора DECLARE**

```
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 3.7. 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' ;
EXECUTE BLOCK AS
BEGIN
   EXCEPTION ex1 USING ('the text');
END !
```

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 3.8. Синтаксис оператора EXIT**

```
EXIT;
```

См. также операторы LEAVE, SUSPEND, CONTINUE.

# FETCH

Оператор FETCH читает очередную запись набора данных, связанного с курсором. Синтаксис оператора:

**Листинг 3.9. 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 предло-

жением 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 3.10. Синтаксис оператора FOR EXECUTE STATEMENT**

```
[FOR] EXECUTE STATEMENT <строковое выражение>
   [ON EXTERNAL [DATA SOURCE] <строка соединения> [READ ONLY | READ WRITE]]
   [WITH {AUTONOMOUS TRANSACTION [READ ONLY | READ WRITE] | 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> ::= <имя параметра>:=<выражение>
                            [, <имя параметра>:=<выражение> ...]

<позиционные параметры> ::= <выражение> [, <выражение> ...]
```

Этот оператор принимает «строковое выражение» и выполняет его, как будто это оператор DSQL.

Если оператор возвращает данные, то с помощью предложения INTO их можно передать в локальные переменные. Именам внутренних переменных в этом предложении должны предшествовать символы двоеточия.

Для каждой считанной записи выполняется составной оператор после ключевого слова DO. Цикл повторяется, пока не будут прочитаны все строки (или пока не встретится оператор LEAVE). После этого происходит выход из цикла.

## Строковое выражение с параметризованным SQL запросом

Новые расширения позволяют использовать оператор EXECUTE STATEMENT с динамическими параметрами аналогично параметризованному DSQL-оператору. Текст самого запроса тоже может быть передан в качестве параметра. Параметры могут быть именованными и позиционными (безымянными). Но совместное использование именованных и неименованных параметров невозможно. Значение должно быть присвоено каждому параметру. Для связывания во время выполнения именованных параметров с их значениями используется оператор «:=». Параметры должны быть поме-

щены в круглые скобки при вызове EXECUTE STATEMENT. Именованным параметрам должно предшествовать двоеточие в самом операторе, но не при присвоении значения параметру. Каждый именованный параметр может использоваться в операторе несколько раз, но только один раз при присвоении значения. Передача значений безымянным параметрам должна происходить в том же порядке, в каком они встречаются в тексте запроса. Для позиционных параметров число подставляемых значений должно точно равняться числу параметров в операторе.

**Пример.** Использование именованных входных параметров:

```
EXECUTE BLOCK AS
  DECLARE Request VARCHAR(255);
  BEGIN
   Request = 'INSERT INTO MyTable VALUES (:a, :b, c, :a)';
   EXECUTE STATEMENT (:Request) (a := 3, b := 'MICHAEL', c:= 'Murr');
  END
```

**Пример.** Использование неименованных входных параметров

```
EXECUTE BLOCK AS
  DECLARE Request VARCHAR(255);
  BEGIN
   Request = 'INSERT INTO MyTable VALUES (?, ?, ?, ?)';
   EXECUTE STATEMENT (:Request) (3, 'MICHAEL', 'Murr', 3);
  END
```

## Предложение WITH {AUTONOMOUS|COMMON} TRANSACTION

Необязательное предложение WITH {AUTONOMOUS|COMMON} TRANSACTION позволяет управлять транзакцией, в которой будет выполняться оператор. Режим COMMON используется по умолчанию.

При использовании предложения WITH AUTONOMOUS TRANSACTION оператор всегда будет выполняться в новой транзакции, запущенной с такими же параметрами, как у текущей транзакции. Но можно задать другой режим чтения/записи с помощью опций READ ONLY | READ WRITE. Эта транзакция будет подтверждена, если оператор выполнится без ошибок, или отменена, если во время выполнения оператора возникли ошибки.

При использовании WITH COMMON TRANSACTION при выполнении оператора в контексте текущего соединения будет использована текущая транзакция.

При использовании WITH COMMON TRANSACTION при соединении с внешним источником данных:

- при первом соединении (в контексте текущей транзакции) с внешним источником данных будет запущена новая транзакция с такими же параметрами (уровень изоляции и т.д.), как у текущей транзакции;
- при следующих соединениях (в контексте текущей транзакции) с этим же внешним источником данных будет использована ранее запущенная транзакция.

## Предложение WITH CALLER PRIVILEGES

Добавление необязательного предложения WITH CALLER PRIVILEGES позволяет оператору наследовать привилегии доступа вызвавшей его хранимой процедуры, функции или триггера. Результат будет таким же, как если бы исполняемый оператор был вызван хранимой процедурой или триггером непосредственно (т.е. без оператора EXECUTE STATEMENT).

> Предложение WITH CALLER PRIVILEGES не может использоваться совместно с предложением ON EXTERNAL DATA SOURCE.

## Предложение ON EXTERNAL [DATA SOURCE]

С помощью предложения ON EXTERNAL DATA SOURCE оператор EXECUTE STATEMENT выполняется в отдельном соединении с той же или другой базой данных, возможно даже на другом сервере.

Параметр <строка соединения> представляет собой обычную строку соединения с базой данных и имеет следующий формат:

```
EXECUTE STATEMENT <строковое выражение>
   ON EXTERNAL [DATA SOURCE] <строка соединения> [READ ONLY | READ WRITE]
   [AS USER <имя пользователя>]
   [PASSWORD <пароль>]

<строка соединения> ::= [{ <имя сервера>: | \\<имя сервера>\ }] { <путь к файлу БД>
| <алиас БД> }
```

Если строка подключения имеет значение NULL или '' (пустая строка), предложение ON EXTERNAL считается отсутствующим и оператор выполняется для текущей базы данных.

При использовании предложения ON EXTERNAL [DATA SOURCE], параметр <строка соединения> которого содержит значение отличное от NULL, оператор всегда будет выполняться в отдельном соединении. Кроме того, внешнее соединение будет установлено и при выполнении оператора EXECUTE STATEMENT с использованием предложения AS USER, параметр <имя пользователя> которого отличается от CURRENT_USER. Установленное внешнее соединение будет существовать до тех пор, пока существует хотя бы одна транзакция, использующая его, и закрывается независимо от способа завершения последней транзакции. Если последняя транзакция, использующая внешнее соединение, подтверждается с помощью CommitRetaining или отменяется с помощью RollbackRetaining, то внешнее соединение не закрывается.

По умолчанию на внешней базе запускается транзакция с такими же параметрами, как у основной (из которой вызывается EXECUTE STATEMENT). С помощью предложений READ ONLY|READ WRITE можно задать другой режим чтения/записи для внешней транзакции.

При выполнении оператора в отдельном соединении используется пул соединений и пул транзакций:

- 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. При использовании предложения WITH COMMON TRANSACTION транзакции будут снова использованы как можно дольше. Они будут подтверждаться или откатываться вместе с текущей транзакцией.
- Внешние соединения, созданные с использованием предложения WITH AUTONOMOUS TRANSACTION, закрываются после выполнения оператора. При использовании предложения WITH AUTONOMOUS TRANSACTION всегда запускается новая транзакция. Она будет подтверждена или отменена сразу же после выполнения оператора.
- 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.

Внешние соединения с одной и той же базой данных на одном и том же сервере, но с разной строкой соединения (параметром предложения ON EXTERNAL), считаются разными соединениями – т.е. при использовании разных протоколов, портов, имен или IP-адресов сервера, а также алиасов баз данных – будут установлены отдельные внешние соединения. При соединении с внешним источником с одной и той же строкой соединения, но под разными учетными записями (AS USER), также будут созданы отдельные внешние соединения.

Если внешний источник данных находится на другом сервере, то предложения AS USER <имя пользователя> и PASSWORD <пароль> являются обязательными, а предложение ROLE <роль> явля-

ется опциональным.

Взаимодействие предложения `ON EXTERNAL` с `AS USER, PASSWORD, ROLE, CERTIFICATE`:

- Если присутствует хотя бы один из параметров `AS USER, PASSWORD` или `ROLE`, то будет предпринята попытка аутентификации с помощью разрешенных плагинов с указанными значениями параметров. Для недостающих параметров не используются никаких значений по умолчанию.

  Значения имени пользователя и пароля передаются в открытой форме, что небезопасно. Например, если ESOE (сокр. от `EXECUTE STATEMENT ON EXTERNAL`) вызывается из кода хранимой процедуры, подключенные пользователи могут видеть логин и пароль. Для безопасного подключения в Ред Базе Данных был разработан плагин аутентификации `ExtAuth` специально для ESOE, который устанавливает доверительную связь между серверами Ред Базы Данных и выполняет аутентификацию ESOE без логина и пароля:

  ```
  EXECUTE STATEMENT 'SELECT * FROM RDB$DATABASE'
  ON EXTERNAL 'server:db1' AS USER 'MYUSER';
  ```

  Как установить и настроить плагин `ExtAuth` описано в Руководстве Администратора.

- 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;
- Если все три параметра отсутствуют, а строка подключения содержит имя сервера (или IP адреса), то будет предпринята попытка доверенной (ExtAuth или Win_Sspi) авторизации к удаленному серверу.
- Если присутствует параметры `CERTIFICATE` и `PIN` (необязательный), то будет предпринята попытка многофакторной (Multifactor) аутентификации с указанными значениями параметров.

*Замечания:*

- 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`.
- Набор символов, используемый для внешнего соединения, совпадает с используемым набором для текущего соединения.
- Двухфазные транзакции не поддерживаются.

## Предложение `AS USER, PASSWORD, ROLE`

Необязательные предложения `AS USER, PASSWORD` и `ROLE` позволяют указывать от имени какого пользователя, и с какой ролью будет выполняться SQL оператор.

Если задано предложения `ON EXTERNAL`, то все случаи присутствия параметров `AS USER, PASSWORD` и `ROLE` рассмотрены в подразделе «Предложение `ON EXTERNAL [DATA SOURCE]`».

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 принимает 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.

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 3.11. Синтаксис оператора FOR SELECT**

```
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

Оператор используется для выполнения ветвления процесса обработки данных в PSQL. Его синтаксис:

**Листинг 3.12. 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 3.13. 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 3.14. 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 3.15. 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 по-

сле этого оператор FETCH (этот оператор неявно осуществляется при выполнении оператора SELECT), работа процедуры возобновляется с оператора, следующего непосредственно за оператором SUSPEND.

**Листинг 3.16. Синтаксис оператора SUSPEND**

```
SUSPEND;
```

Если в селективной процедуре возникает ошибка, инструкции, выполненные после последнего оператора SUSPEND будут отменены. Операторы, выполненные до последнего SUSPEND не будут отменены, если транзакция не будет отменена.

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 3.17. 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 3.18. Синтаксис оператора WHILE-DO**

```
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().