РЕД База Данных

Версия 5

Внешние хранимые процедуры, функции и триггеры, написанные на языке Java

Содержание

1 Особенности написания внешних процедур, функций и триггеров 3
1.1 Взаимодействие программ базы данных и Java методов . . . 4
1.1.1 Соответствие типов данных SQL и Java . . . 4
1.1.2 Доступ к контексту функции, процедуры и триггера из метода на Java . . . 5
1.1.3 Доступ к контексту вызова из Java метода . . . 8
1.2 Безопасность . . . 9
1.3 Правила написания тела внешних процедур, функций и триггеров на Java . . . 10

2 Работа с внешними хранимыми процедурами, функциями и триггерами 14
2.1 Объявление/изменение/пересоздание внешних процедур . . . 14
2.1.1 Примеры . . . 14
2.2 Объявление/изменение/пересоздание внешних функций . . . 15
2.2.1 Примеры . . . 16
2.3 Объявление/изменение/пересоздание внешних триггеров . . . 16
2.3.1 Примеры . . . 17
2.4 Удаление внешних процедур, функций и триггеров . . . 17
2.5 Вызов внешних процедур и функций . . . 18

3 Примеры внешних процедур, функций и триггеров 19
3.1 Вычисление факториала числа . . . 19
3.2 Пример работы с файловой системой . . . 19
3.3 Пример внешней процедуры, возвращающей набор данных . . . 20
3.4 Пример внешней процедуры, осуществляющей работу с другой базой данных . . . 21

Глава 1

Особенности написания внешних процедур, функций и триггеров

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.

Внешние процедуры, функции или триггеры могут располагаться в jar-файлах. В РЕД База Данных для работы с этими файлами используется плагин `JavaEngine`, который позволяет запускать функции, процедуры и триггеры на платформе Java.

Для использования описываемого в этом руководстве функционала необходимо установить JRE не ниже 11, а также настроить параметры взаимодействия сервера «РЕД База Данных» с виртуальной машиной Java.

Для этого в файле конфигурации `plugins.conf`, который расположен в корневом каталоге установки сервера, необходимо раскомментировать секции относящиеся к `JAVA` (`Plugin = JAVA` и `Config = JAVA_config`). Внутренние системные классы, необходимые для загрузки JavaEngine, по умолчанию находятся в папке `/jar` установки сервера. Путь к ним настраивается параметром `JarDirs`, где можно указать несколько директорий, разделенных : в linux, ; в Windows.

В файле конфигурации `firebird.conf` в параметре `JavaHome` укажите путь к установленному JDK или JRE.

```
JavaHome = /usr/lib/jvm/java-11-openjdk-amd64
```

Начиная с версии Java 18 в `jvm.args` нужно использовать настройку:

```
-Djava.security.manager=allow
```

Установка этого свойства позволяет разрешить динамическую установку Security Manager. Это необходимо, так как начиная с Java 18 по умолчанию это свойство имеет значение `disallow`, и с JavaEngine может возникнуть ошибка.

Пользовательские классы, содержащие методы, которые будут использоваться в качестве тела внешних процедур, функций и триггеров, должны храниться в виде jar-файлов там, куда указывает значение параметра `Classpath` файла конфигурации `firebird.conf`.

```
# Настройка пути к jar файлам для всех баз данных:
Classpath = ["/path/to/jars", "myjar"]
```

Пути до jar-файлов можно указать отдельно для каждой базы данных (в `databases.conf`), либо для всех баз одновременно (в `firebird.conf`). Значения указанные в `Classpath` в `databases.conf` добавляются к значениям указанным в `firebird.conf`. Если список jar для какой-либо базы данных нужно полностью заменить, то нужно использовать опцию `OverrideClasspath` в `databases.conf`:

```
# Настройка пути к jar файлам для конкретной базы данных:
employee = $(dir_sampleDb)/employee.fdb
{
       OverrideClasspath = ["jar_emp"]
}
```

Классы в файловой системе доступны всем базам данных, обрабатываемыми процессом RDB. По аналогии с сервером приложений они являются системными классами.

1.1 Взаимодействие программ базы данных и Java методов

Взаимодействие с Java методами из базы данных происходит путем объявления их в базе данных с указанием места расположения во внешнем модуле с помощью предложения EXTERNAL NAME. Аргументом этого предложения является строка, в которой через разделитель указано имя внешнего модуля, имя программы внутри модуля и определённая пользователем информация. В предложении ENGINE указывается имя движка для обработки подключения внешних модулей. В данном случае это JAVA.

```
EXTERNAL NAME '<полное имя класса>.<имя static метода>!(
              [<Java тип> [, <Java тип>...]])'
              [!<определяемая пользователем информация>]
ENGINE JAVA
```

Существует два основных способа сопоставить функции и процедуры базы данных с методами Java. Это фиксированные и обобщенные сигнатуры. Триггеры могут отображаться только с помощью обобщенных сигнатур.

Фиксированные сигнатуры означают, что для каждого параметра программы (функции, процедуры) базы данных должен быть параметр в Java методе.

Обобщенные сигнатуры не имеют параметров. Java-код может, с помощью интерфейса контекстов (Context, Values и др.), получить все параметры или значения полей, переданные программой базы данных.

1.1.1 Соответствие типов данных SQL и Java

Плагин JavaEngine поддерживает типы Java, представленные в *таблице*.

Таблица 1.1 — Поддерживаемые Java типы

Java тип	Совместимый SQL тип
byte[]	BLOB, CHAR, VARCHAR
boolean	любой
short	любой
int	любой
long	любой*
float	любой*
double	любой*
java.lang.Boolean	любой
java.lang.Short	любой
java.lang.Integer	любой
java.lang.Long	любой
java.lang.Float	любой

(разрыв таблицы)

(разрыв таблицы)

Java тип	Совместимый SQL тип
java.lang.Double	любой
java.lang.Object	любой**
java.lang.String	любой
java.math.BigDecimal	любой
java.sql.Blob	BLOB
org.firebirdsql.jdbc.FirebirdBlob	BLOB
java.sql.Time	любой
java.sql.Timestamp	любой
java.util.Date	любой

* - SQL значение NULL конвертируется в 0 примитивного числового типа Java или в false типа boolean.

** - Сопоставление значений типа Object и SQL типов происходит в соответствии с представленной ниже таблицей.

Таблица 1.2 — Сопоставление типов SQL и Java

Тип SQL	Тип Java
NUMERIC	java.math.BigDecimal
DECIMAL	java.math.BigDecimal
SMALLINT	java.math.BigDecimal
INTEGER	java.math.BigDecimal
BIGINT	java.math.BigDecimal
FLOAT	java.lang.Float
DOUBLE PRECISION	java.lang.Double
CHAR	java.lang.String
VARCHAR	java.lang.String
BLOB	java.sql.Blob
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp
BOOLEAN	java.lang.Boolean

1.1.2 Доступ к контексту функции, процедуры и триггера из метода на Java

Для доступа к контексту функций, процедур и триггеров реализовано несколько интерфейсов:

- Context - отражает информацию о функциях, процедурах и триггерах базы данных.
- CallableRoutineContext - отражает контекст внешних процедур и функций.
  Наследуется от Context.
- FunctionContext - отражает контекст внешних функций.

Наследуется от `CallableRoutineContext`.

- `ProcedureContext` - отражает контекст внешних процедур.

  Наследуется от `CallableRoutineContext`.
- `TriggerContext` - отражает контекст внешних триггеров.

  Наследуется от `Context`.
- `ExternalResultSet` - представляет `ResultSet` для внешних хранимых процедур выбора.
- `Values` - позволяет получать или устанавливать значения параметров у функций или процедур и полей у триггеров.
- `ValuesMetadata` - позволяет получать значения метаданных параметров функций или процедур и полей триггеров.

  Наследуется от `java.sql.ParameterMetaData`.
- `TriggerContext.Action` - перечисление (`enum`) для операций, вызвавших триггер.

  Наследуется от `java.lang.Enum`.
- `TriggerContext.Type` - перечисление (`enum`) для типа триггера.

  Наследуется от `java.lang.Enum`.

Интерфейсы доступа к контексту функций, процедур и триггеров:

Таблица 1.3 — Context

Имя метода	Тип	Описание
`get()`	`static Context`	Получение экземпляра объекта Context, связанного с текущим вызовом.
`getBody()`	`String`	Получение исходного кода процедуры, функции или триггера. Всегда будет пустым.
`getConnection()`	`Connection`	Получение объекта класса `Connection`. Существует и другой способ: `DriverManager.getConnection ("jdbc:default:connection")`
`getNameInfo()`	`String`	Получение определяемой пользователем информации для передачи в тело внешней функции, процедуры, триггера.
`getObjectName()`	`String`	Получение имени объекта метаданных, вызвавшего внешнюю процедуру, функцию или триггер.

Таблица 1.4 — CallableRoutineContext

Имя метода	Тип	Описание
`get()`	`static CallableRoutineContext`	Получение экземпляра объекта `CallableRoutineContext`, связанного с текущим вызовом.
`getInputMetadata()`	`ValuesMetadata`	Получение метаданных входных параметров.
`getInputValues()`	`Values`	Получение значений входных параметров.

(разрыв таблицы)

(разрыв таблицы)

Имя метода	Тип	Описание
`getOutputMetadata()`	`ValuesMetadata`	Получение метаданных выходных параметров.
`getPackageName()`	`String`	Получение имени пакета базы данных, из которого вызвана внешняя функция или процедура.

Таблица 1.5 — FunctionContext

Имя метода	Тип	Описание
`get()`	`static ProcedureContext`	Получение экземпляра объекта `ProcedureContext`, связанного с текущим вызовом.
`getOutputValues()`	`Values`	Получение значений выходных параметров.

Таблица 1.6 — TriggerContext

Имя метода	Тип	Описание
`get()`	`static TriggerContext`	Получение экземпляра объекта `TriggerContext`, связанного с текущим вызовом.
`getAction()`	`TriggerContext.Action`	Получение операции, вызвавшей триггер. Возможные значения: `CONNECT`, `DISCONNECT`, `TRANS_COMMIT`, `TRANS_ROLLBACK`, `TRANS_START`, `INSERT`, `UPDATE`, `DELETE`, `DDL`.
`getFieldsMetadata()`	`ValuesMetadata`	Получение метаданных полей триггера.
`getNewValues()`	`Values`	Получение новых значений полей.
`getOldValues()`	`Values`	Получение старых значений полей.
`getTableName()`	`String`	Получение имени таблицы, для которой вызван триггер.
`getType()`	`TriggerContext.Type`	Получение типа триггера. Возможные значения: `AFTER`, `BEFORE`, `DATABASE`.

Таблица 1.7 — ExternalResultSet

Имя метода	Тип	Описание
`get()`	`static TriggerContext`	Получение экземпляра объекта `TriggerContext`, связанного с текущим вызовом.
`close()`	`default void`	Вызывается RDB после метода `fetch()`.
`fetch()`	`boolean`	Вызывается `RDB` для получения набора записей `ExternalResultSet`.

Таблица 1.8 — Values

Имя метода	Тип	Описание
`get()`	`static TriggerContext`	Получение экземпляра объекта `TriggerContext`, связанного с текущим вызовом.
`getObject(int index)`	`Object`	Получение значения в виде объекта по данному индексу.
`setObject(int index, Object value)`	`Object`	Устанавливается значение объекта по данному индексу.

Таблица 1.9 — ValuesMetadata

Имя метода	Тип	Описание
`getIndex(String name)`	`int`	Получение индекса по заданному имени.
`getJavaClass(int index)`	`Class<?>.`	Получение Java класса по данному индексу.
`getName(int index)`	`String`	Получение имени по данному индексу.

Более подробную информацию обо всех интерфейсах и их методах можно найти здесь.

1.1.3 Доступ к контексту вызова из Java метода

В "РЕД База Данных" имеется возможность получения доступа к контексту выполнения, откуда была вызвана внешняя процедура, функция или триггер.

Получить доступ к контексту вызова из метода на `Java` можно путем вызова метода `getConnection()` интерфейса `Context`.

Существует и другой способ. В соответствии со стандартом `SQLJ`, можно установить соединение, используется метод `DriverManager.getConnection()`. Для этого необходимо использовать `URL` вида `jdbc:default:connection:`.

```
Connection connection =
DriverManager.getConnection("jdbc:default:connection:");
```

В этом случае все вызовы `connection.commit()` и `connection.rollback()` будут проигнорированы.

Кроме того, в дополнении к стандарту `SQLJ`, в "РЕД База Данных" можно получить доступ из `java`-метода к базе данных с отдельной транзакцией. Для этого используется `URL` вида `jdbc:new:connection:`.

```
Connection connection =
DriverManager.getConnection("jdbc:new:connection:");
```

Это позволит, например, выполнить `DDL` или другую операцию, требующую автономности для выполнения.

1.2 Безопасность

Механизм java-security устарел и будет удален в РЕД Базе Данных 6.

Одной из наиболее важных особенностей платформы Java является система безопасности, так называемая "песочница". JavaEngine интегрирует механизм безопасности J2SE/JAAS с РЕД Базой Данных, так что права могут быть назначены пользователям базы данных, использующим код Java.

Права доступа пользователей действуют на уровне сервера. Они хранятся в базе данных безопасности java-security.fdb. Эта база содержит следующие таблицы:

Таблицы	Поля	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 %).

CLASS_NAME	ARG1	ARG2
java.util.PropertyPermission	file.separator	read
java.util.PropertyPermission	java.version	read
java.util.PropertyPermission	java.vendor	read

(разрыв таблицы)

(разрыв таблицы)

CLASS_NAME	ARG1	ARG2
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

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')
```

1.3 Правила написания тела внешних процедур, функций и триггеров на Java

1. В качестве тела внешний программ, написанных на языке Java, может использоваться любой метод, объявленный как public static, и принадлежащий к public-классу (не внутреннему inner-классу).

```
package example;
public class ExampleClass {
   public static void proc1() {
      //operators on Java
   }
   ...
}
```

2. При написании тела внешних процедур на Java, допускается использование в качестве выходных параметров только типы данных ExternalResultSet (или класс, реализующий этот интерфейс), ResultSet и void. В качестве входных параметров возможно использование любых типов данных и их комбинаций, описанных в разделе *Соответствие типов данных SQL и Java*.

```
public static void proc1() throws SQLException {
    Connection con = DriverManager.getConnection("jdbc:new:connection:");
    PreparedStatement pstmt;
    pstmt = con.prepareStatement("INSERT INTO test_table values (12)");
    pstmt.execute();
    pstmt.close();
}
public static ExternalResultSet proc2(int i, int[] o) {
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
    o[0] = i;
    return new ExternalResultSet() {
       @Override
       public boolean fetch() throws Exception {
          return ++o[0] <= i + 5;
       }
    };
}
```

3. Набор данных `ResultSet` может быть получен как результат внутреннего запроса, запроса к внешней базе данных или другим способом. Например, `ResultSet` может быть вычислен при помощи какого-либо алгоритма.

```
public static ResultSet proc3() throws SQLException {
    Connection con = DriverManager.getConnection("jdbc:new:connection:");
    PreparedStatement pstmt;
    pstmt = con.prepareStatement("SELECT \* FROM test_table");
    return pstmt.executeQuery();
}
```

4. При написании внешних функций на Java методы могут возвращать результат любого допустимого типа. В качестве входных параметров возможно использование любых типов данных и их комбинаций, описанных в разделе *Соответствие типов данных SQL и Java*.

```
public static String func1(java.math.BigDecimal i, Short j, short k, String l,
java.sql.Date m, Timestamp n) {
    String z = i.toString() + "_" + j + "_" + k + "_" + l + "_" + m + "_" + n;
    return z;
}
```

5. Метод, реализующий тело внешнего триггера, не должен содержать входных параметров. Тип возвращаемого результата - `void`.

```
public static void trigger1() throws SQLException {
    TriggerContext context = TriggerContext.get();
    context.getNewValues().setObject(1, "It is new value from Java");
}
```

6. Методы, реализующие тела внешних процедур и функций, либо не имеют входных параметров. В таком случае используется доступ к контексту с получением всех параметров, переданных процедурой или функцией.

```
create function funcSum4 (n1 integer, n2 integer, n3 integer, n4 integer)
returns integer
external name 'example.ExampleClass.sumFunc()'
engine java;

public static Integer sumFunc() {
   FunctionContext context = FunctionContext.get();
   ValuesMetadata inputM = context.getInputMetadata();
   Values input = context.getInputValues();
   Integer result = 0;
   for(int i = 1; i <= inputM.getParameterCount(); i = i+1)
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
    result = result + input.getObject(i);
    return result;
}
```

Либо количество и типы параметров метода на Java соответствуют количеству и типам объявленной в базе данных процедуры или функции.

```
create function extFunc1 (n1 integer, n2 smallint, n3 smallint, n4
varchar(10), n5 date, n6 timestamp)
returns varchar(100)
external name 'example.ExampleClass.func1(java.math.BigDecimal, Short,short,
String, java.sql.Date, Timestamp)'
engine java;
```

7. Выходные параметры процедуры (если такие имеются) должны отображаться в спецификации вызова в виде массивов. `JavaEngine` принимает каждый выходной параметр в виде массива длиной 1, а процедуры могут изменять свой [0] элемент.

```
create procedure testProc4 (numRows integer) returns (n integer)
external name 'example.ExampleClass.proc4(int, int[])'
engine java;

create procedure testProc2 (numRows integer) returns (n integer)
external name 'example.ExampleClass.proc2(int, int[])'
engine java;

public static void proc4(int i, int[] o) {
    o[0] = i;
}
```

8. Во внешних табличных триггерах вычисляются все значения полей таблицы, независимо от того есть ли на них ссылка в теле триггера. В некоторых случаях, например для "тяжелых" вычисляемых (`COMPUTED BY`) столбцов, стоит отключить их расчет, если они не используются в триггере. Для этого можно использовать следующие аннотации к функциям триггера:
    - `@DoNotEvaluateComputedFields` — не вычислять все `COMPUTED BY` столбцы;
    - `@DoNotEvaluateField(name = "<имя столбца>")` — не вычислять столбец с конкретным именем;
    - `@EvaluateField(name = "<имя столбца>")` — вычислить только столбец с указанным именем.

Аннотацию `@EvaluateField` нельзя использовать совместно с двумя другими, т.к. она исключает их множество столбцов. Иначе будет выдано сообщение об ошибке.

```
CREATE OR ALTER TRIGGER TEST_TRIGGER
FOR TEST_TABLE
AFTER INSERT
external name esp.TestTrigger.evaluate_field_insert()
engine JAVA;

@DoNotEvaluateComputedFields
@DoNotEvaluateField(name = "TEST2")
private static void evaluate_field_insert() throws SQLException {
    Trigger trigger = new Trigger();
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
    Object test1New = trigger.getObject_New("TEST1");
    Object test3New = trigger.getObject_New("TEST3");
    try (Connection con=DriverManager.getConnection ("...")) {
       try (PreparedStatement pstmt = con.prepareStatement("INSERT INTO test_
table2 values (?, ?)")) {
          pstmt.setString(1, (String) test1New);
          pstmt.setString(2, (String) test3New);
          pstmt.execute();
      }
   }
}
```

Перегрузка методов разрешена, но её следует избегать, потому что, например, методы с параметрами `int` и `Integer` различаются с точки зрения Java, но не с точки зрения внешних процедур и функций "РЕД База Данных".

Глава 2

Работа с внешними хранимыми процедурами, функциями и триггерами

2.1 Объявление/изменение/пересоздание внешних процедур

Синтаксис операторов создания, изменения и пересоздания внешней хранимой процедуры, написанной на Java, имеет одинаковую структуру и приведен в листинге:

```
{CREATE [ OR ALTER ] | RECREATE | ALTER} PROCEDURE <имя хранимой процедуры>
[AUTHID {OWNER | CALLER}]
   [(<входной параметр> [, <входной параметр> ...])]
[RETURNS (<выходной параметр> [, <выходной параметр> ...])]
[SQL SECURITY {DEFINER | INVOKER}]
EXTERNAL NAME '<полное имя класса>.<имя static метода>!(
              [<Java тип> [, <Java тип>...]])'
              [!<определяемая пользователем информация>]
ENGINE JAVA
```

Внешние процедуры либо не имеют выходных параметров , либо возвращают набор данных `ExternalResultSet` (или класс, реализующий этот интерфейс), в зависимости от того, является ли процедура селективной или нет. Выходные параметры при вызове функций должны быть массивами. JavaEngine передает каждый параметр как массив с длиной 1, таким образом процедуры могут менять их нулевой элемент.

2.1.1 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);
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
            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;
       }
    };
}
```

2.2 Объявление/изменение/пересоздание внешних функций

Синтаксис операторов создания, изменения и пересоздания внешней функции, написанной на Java, имеет одинаковую семантику и приведен в листинге:

```
{CREATE [ OR ALTER ] | RECREATE | ALTER} FUNCTION <имя функции>
[AUTHID {OWNER | CALLER}]
   [(<входной параметр> [, <входной параметр> ...])]
RETURNS (<тип данных>)
[SQL SECURITY {DEFINER | INVOKER}]
EXTERNAL NAME '<полное имя класса>.<имя static метода>!(
               [<Java тип> [, <Java тип>...]])'
               [!<определяемая пользователем информация>]
ENGINE JAVA
```

В отличие от внешних процедур, внешние функции всегда возвращают одно значение какого-либо из типов описанных в разделе *Соответствие типов данных SQL и Java*.

2.2.1 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;
```

2.3 Объявление/изменение/пересоздание внешних триггеров

Синтаксис операторов создания, изменения и пересоздания внешнего триггера, написанного на Java, имеет одинаковую семантику и приведен в листинге:

```
{CREATE [ OR ALTER ] | RECREATE | ALTER} TRIGGER <имя триггера>
[AUTHID {OWNER | CALLER}]
{
<объявление табличного триггера>
 | <объявление табличного триггера в стандарте SQL-2003>
 | <объявление триггера базы данных>
 | <объявление DDL триггера> }
[SQL SECURITY {DEFINER | INVOKER}]
EXTERNAL NAME '<полное имя класса>.<имя static метода>!(
               [<Java тип> [, <Java тип>...]])'
               [!<определяемая пользователем информация>]
ENGINE JAVA
```

Спецификация вызовов триггеров всегда без параметров, и Java метод должен возвращать `void`. Детали вызова и значения полей `OLD` и `NEW` можно получить и установить из контекста вызова.

2.3.1 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;
```

2.4 Удаление внешних процедур, функций и триггеров

Удаление внешней процедуры осуществляется оператором `DROP PROCEDURE`.

```
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:

```
DROP FUNCTION <имя_функции>;
```

Оператор `DROP TRIGGER` удаляет существующий триггер:

```
DROP TRIGGER <имя_триггера>;
```

2.5 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 3

Примеры внешних процедур, функций и триггеров

3.1 Вычисление факториала числа

В *примере* приведено описание внешней функции, написанной на Java, которая производит рекурсивное вычисление факториала числа:

```
public static long factorial(int x) throws SQLException {
    Connection con = DriverManager.getConnection(jdbc:default:connection:);
    PreparedStatement pstmt = con.prepareStatement("select factor(?) from rdb$database
");
    if ((x == 0) \|\| (x == 1)) {
       return 1;
    }
    else {
       pstmt.setLong(1, x - 1);
       ResultSet rs = pstmt.executeQuery();
       rs.next();
       return x \* rs.getLong(1);
    }
}
```

Регистрация в базе данных функции из листинга:

```
CREATE FUNCTION factor (x integer)
RETURNS bigint
EXTERNAL NAME 'esp.TestESP.factorial(int)';
ENGINE JAVA;
```

Результатом выполнения запроса

```
SELECT FACTOR(5) FROM RDB$DATABASE;
```

будет число 120.

3.2 Пример работы с файловой системой

В следующем листинге приведено описание внешней процедуры, написанной на Java, иллюстрирующей пример работы с файловой системой:

```
public static void loadFile(String path) throws SQLException, IOException {
    InputStream is = new FileInputStream(path);
    byte[] b = new byte[is.available()];
    is.read(b);
    Connection con = DriverManager.getConnection("jdbc:default:connection:");
    PreparedStatement pstmt = con.prepareStatement("INSERT INTO test_table values (?)
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
");
    try {
       pstmt.setBytes(1, b);
       pstmt.execute();
    }
    finally {
       pstmt.close();
    }
}
```

Во внешнюю процедуру передаётся параметр строкового типа, задающий путь к файлу. Внутри процедуры осуществляется считывание этого файла в массив типа `byte[]`, с последующей вставкой в таблицу `test_table`, содержащую поле типа `BLOB`.

Скрипты создания таблицы `TEST_TABLE` и регистрации внешней процедуры для примера из листинга приведены ниже:

```
CREATE TABLE test_table(b blob);
CREATE OR ALTER PROCEDURE test( s char(100) )
EXTERNAL NAME 'esp.TestESP.loadFile(String)'
ENGINE JAVA;
```

Пример SQL-запроса выполнения внешней процедуры:

```
EXECUTE PROCEDURE TEST('c:\\image.jpg')
```

3.3 Пример внешней процедуры, возвращающей набор данных

В следующем листинге приведено описание внешней процедуры, написанной на Java, возвращающей набор данных:

```
public static ExternalResultSet testRS(final String property, final String[] result) {
    return new ExternalResultSet() {
       boolean first = true;
       public boolean fetch() throws Exception {
          if(this.first) {
             result[0] = System.getProperty(property);
             this.first = false;
             return true;
          } else {
             return false;
          }
       }
    };
}
```

Скрипт регистрации внешней процедуры приведены ниже:

```
CREATE OR ALTER PROCEDURE test(prop char(50))
RETURNS (res char(150))
```

(продолжение на следующей странице)

(продолжение с предыдущей страницы)

```
EXTERNAL NAME 'esp.TestESP.testRS(String, String[])'
ENGINE JAVA;
```

Просмотреть результат работы внешней процедуры `testRS` можно выполнив запрос

```
SELECT * FROM TEST("java.home");
```

3.4 Пример внешней процедуры, осуществляющей работу с другой базой данных

С помощью внешних процедур возможны подключения к другим базам данных посредством JDBC. Пример тела внешней процедуры, осуществляющей подключение к другой БД, делающей в ней выборку данных из таблицы `test_table` и вставляющей выбранные данные в таблицу `test_table` базы данных, из которой была вызвана внешняя процедура, приведен ниже:

```
public static void interConnect() throws SQLException {
    Connection con = DriverManager.getConnection("jdbc:new:connection:");
    PreparedStatement pstmt = con.prepareStatement("INSERT INTO test_table values (?)
");
    try {
       String url = "jdbc:firebirdsql:localhost:D:/testbase.fdb";
       Connection extCon = DriverManager.getConnection(url, "SYSDBA", "masterkey");
       PreparedStatement extPstmt = extCon.prepareStatement("SELECT * FROM test_table
");
       try {
          ResultSet rs = extPstmt.executeQuery();
          while (rs.next()) {
             pstmt.setString(1, rs.getString(1));
             pstmt.execute(); }
       }
       finally {
          extPstmt.close(); }
    }
    finally {
       pstmt.close(); }
}
```

Скрипты создания таблицы `test_table` и регистрации внешней процедуры приведены ниже:

```
CREATE TABLE test_table(f_varchar varchar (50))
CREATE OR ALTER PROCEDURE test
EXTERNAL NAME 'esp.TestESP.interConnect()'
ENGINE JAVA;
```

Для вызова внешней процедуры `TEST` следует использовать оператор `EXECUTE PROCEDURE`:

```
EXECUTE PROCEDURE TEST;
```