Statement. Creation & executing.

JDBC 4.0

Выполнение SQL-запросов

В JDBC есть три класса для посылки SQL-запросов в БД и три метода в интерфейсе Connection создают экземпляры этих классов.

Statement - создается методом createStatement. Объект Statement используется при простых SQL-запросах.

PreparedStatement - создается методом prepareStatement. Объект PreparedStatement используется в SQL-запросах с одним или более входными параметрами (IN parameters).

Выполнение SQL-запросов

CallableStatement - создается методом prepareCall. Объекты CallableStatement используются для выполнения т.н. хранимых процедур - именованных групп SQL-запросов, наподобие вызова подпрограммы.

Создание JDBC statement

```
Statement stmt = con.createStatement() ;
```

Создает объект Statement для передачи запросов SQL к базе данных

Выполнение запроса через объект Statement

Интерфейс Statement предоставляет три различных метода выполнения SQL-выражений: executeQuery, executeUpdate и execute, в зависимости от SQL-запроса.

Метод executeQuery необходим для запросов, результатом которых является один единственный набор значений, таких как запросов SELECT.

Метод executeUpdate используется для выполнения операторов INSERT, UPDATE или DELETE, а также для операторов DDL (Data Definition Language - язык определения данных)

executeUpdate возвращает целое число, показывающее, сколько строк было модифицировано.

Для выражений типа CREATE TABLE и DROP TABLE, которые не оперируют над строками, возвращаемое методом executeUpdate значение всегда равно нулю.

Транзакции и JDBC

- JDBC позволяет группировку выражений SQL в единую транзакцию.
- Управление транзакциями происходит через объект Connection, с установленным по умолчанию режимом auto-commit
- Режим auto-commit выключается при помощи setAutoCommit(false);
- Режим auto-commit включается при помощи setAutoCommit(true);
- Когда режим auto-commit выключен, все операции модификации данных не записываются до вызова commit();
- После этого все изменения записываются в базу.

Метод execute используется, когда операторы SQL возвращают более одного набора данных, более одного счетчика обновлений или и то, и другое

Закрытие объектов Statement

Объекты **Statement** закрываются автоматически с помощью сборщика мусора виртуальной машины Java.

Statement можно закрыть при помощи close

Закрытие объектов Statement сразу же освобождает ресурсы СУБД и позволяет избежать проблем с памятью.

Подстановочный (escape) синтаксис SQL в объектах Statement

Escape-конструкция заключается в фигурные скобки и ключевое слово:
{ключ.слово . . . параметры . . . }

escape для эскейп-последовательности операции LIKE

{escape 'escape-character'}

stmt.executeQuery("SELECT name FROM Identifiers WHERE Id LIKE `\_%' {escape `\'};

fn для скалярных функций

{fn concat("Hot", "Java")};

SELECT {fn curdate() }
SELECT {fn curtime() }
SELECT {fn now() }

Использование метода execute

Метод execute должен использоваться тогда, когда возможно возвращение нескольких объектов ResultSet, более одного счетчика обновлений или комбинации объектов ResultSet и счетчиков обновлений

Такие множественные результаты, хоть и редки, но возможны при вызове некоторых хранимых процедур или динамическом вызове неизвестного на этапе компиляции SQL-запроса.

После вызова метода execute и выполнения процедуры необходимо вызвать метод getResultSet для получения первого набора данных.

Чтобы получить следующий набор возвращенных данных надо вызвать getMoreResults и затем getResultSet второй раз.

Метод execute возвращает true, если результатом является объект ResultSet, и false, если int.

Если результат SQL-запроса - это не набор данных, то метод getResultSet возвратит null.

Выполнение запросов

Оператор (statement) называется завершенным (complete), если он выполнился и все его результаты были возвращены.

executeQuery - оператор завершен, если считаны все строки соответствующего объекта ResultSet, который был возвращен методом executeXXX.

executeUpdate - оператор завершен сразу после выполнения оператора.

execute оператор остается не завершенным до тех пор, пока все наборы данных или счетчики обновлений, сгенерированные оператором, не будут считаны.

Обработка ошибок JDBC

Когда в JDBC происходит ошибка выбрасывается исключение **SQLException.** В нем содержится следующая информация:

- Описание ошибки. Строку содержащую ошибку можно получить при помощи метода getMessage объекта SQLException..
- Код состояния SQLState. Эти коды стандартизированы ISO/ANSI и Open Group (X/Open). Код ошибки это строка состоящая из 5 символов которую можно получить вызовом getSQLState.
- Код ошибки. Зависит от производителя и указывает на код ошибки которая привела к SQLException. Получается путем вызова getErrorCode.
- Причина. SQLException может содержать ряд объектов типа Throwable которые содержат информацию о причинах вызова SQLException. Возжможно получить при помощи вызова getCause до тех пора не будет получено значение null.
- Ссылка на цепочку исключений доступ к которой происходит при помощи getNextException.

ResultSet

Обработка результатов выполнения запроса производится методами интерфейса **ResultSet**

ResultSet содержит все строки, удовлетворяющие условиям в SQL-выражении и предоставляет доступ к данным в этих строках посредством набора get-методов, которые организуют доступ к колонкам текущей строки.

Метод ResultSet.next используется для перемещения к следующей строке ResultSet, делая ее текущей.

Строки и курсоры

- ResultSet содержит курсор, который указывает на текущую строку данных.
- Каждый раз, когда выполняется метод next, курсор перемещается на одну строку вниз.
- Изначально курсор спозиционирован перед первой строкой, и первый вызов next премещает его на первую строку (она становится текущей).
- С каждым успешным вызовом next курсор перемещается вниз на одну строку, начиная с самой верхней в ResultSet.

Колонки

Методы getXXX предоставляют доступ к значениям в колонках в текущей строке.

В пределах одной строки значения могут быть считаны в любом порядке, но ради обеспечения большей совместимости рекомендуется считывать их подряд слева направо и делать это только один раз.

Колонки

Для указания колонки можно использовать либо ее имя, либо ее номер.

```
String s = rs.getString("title");
String s = rs.getString(2);
```

Значения NULL в результатах

Значение возвращаемое ResultSet.getXXX, равно:

- null для тех из методов getXXX, которые возвращают объекты (такие методы, как getString, getBigDecimal, getBytes, getDate, getTime, getTimestamp, getAsciiStream, getUnicodeStream, getBinaryStream, getObject).

- нулевое значение для getByte, getShort, getInt, getLong, getFloat, and getDouble.

- false в случае getBoolean.

PreparedStatement

Интерфейс PreparedStatement наследуется от Statement и отличается следующим:

- Экземпляры PreparedStatement "помнят" скомпилированные SQL-выражения.
- SQL-выражения в PreparedStatement могут иметь один или более входной (IN) параметр. Вместо него в выражении на месте каждого входного параметра ставится знак ("?").

Создание объектов PreparedStatement

```
PreparedStatement pstmt = con.prepareStatement(
    "UPDATE table4 SET m = ? WHERE x = ?");
```

Передача входных (IN) параметров

Перед выполнением объекта PreparedStatement надо установить значения всех его параметров. Это делается с помощью методов setXXX.

```
pstmt.setLong(1, 123456789);
pstmt.setLong(2, 100000000);
```

```
pstmt.setString(1, "Hi");
for (int i = 0; i < 10; i++) {
    pstmt.setInt(2, i);
    int rowCount = pstmt.executeUpdate();
}
```

Использование объектов

Возможно явно задать конвертирование входных параметров в определенный JDBC-тип с помощью метода setObject

Если JDBC-тип не задан, то драйвер просто преобразует Object в его ближайший JDBC-эквивалент

Отображение JDBC-типов в Java-типы

JDBC type	Java type
CHAR	String
VARCHAR	String
LONGVARCHAR	String
NUMERIC	java.math.BigDecimal
DECIMAL	java.math.BigDecimal
BIT	boolean
TINYINT	byte
SMALLINT	short

INTEGER	int
BIGINT	long
REAL	float
FLOAT	double
DOUBLE	double
BINARY	byte[]
VARBINARY	byte[]
LONGVARBINARY	byte[]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp

Отображение Java-типов в JDBC-типы

Java-тип	JDBC-тип
String	VARCHAR or LONGVARCHAR
java.math.BigDecimal	NUMERIC
boolean	BIT
byte	TINYINT
short	SMALLINT
int	INTEGER
long	BIGINT
float	REAL
double	DOUBLE
byte[]	VARBINARY or LONGVARBINARY
java.sql.Date	DATE
java.sql.Time	TIME
java.sql.Timestamp	TIMESTAMP

Отображение JDBC-типов на объектные типы Java

JDBC Type	Java Object Type
CHAR	String
VARCHAR	String
LONGVARCHAR	String
NUMERIC	java.math.BigDecimal
DECIMAL	java.math.BigDecimal
BIT	Boolean
TINYINT	Integer
SMALLINT	Integer
INTEGER	Integer
BIGINT	Long
REAL	Float
FLOAT	Double
DOUBLE	Double
BINARY	byte[]
VARBINARY	byte[]
LONGVARBINARY	byte[]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp

Отсылка значений NULL в качестве входного параметра

Метод setNull позволяет отсылать значения NULL в БД как входные параметры.

JDBC-значение NULL будет отослано в БД также в том случае, если методу setXXX будет передано Java-значение null (если метод принимает Java-объект в качестве аргумента).

Отправка очень больших входных параметров

Для посылки неограниченного количества данных используются методы setBytes и setString

При установке входного параметра в значение Java-потока ввода (input stream) появляется возможность передавать данные из потока ввода вывода

```
java.io.File file = new java.io.File("/tmp/data");
int fileLength = file.length();
java.io.InputStream fin = new
    java.io.FileInputStream(file);

java.sql.PreparedStatement pstmt =
    con.prepareStatement( "UPDATE Table5 SET stuff =
    ? WHERE index = 4");

pstmt.setBinaryStream (1, fin, fileLength);
    pstmt.executeUpdate();
```

Класс CallableStatement

Объект CallableStatement предоставляет унифицированный способ вызова хранимых процедур в любой СУБД. Вызов процедуры осуществляется с помощью escape-синтаксиса в одной из двух форм: с результирующим параметром и без него.

Синтаксис вызова хранимой процедуры

```
{call имя_процедуры[(?, ?, ...)]}
```

Синтаксис для процедуры, возвращающей результат:

```
{? = call имя_процедуры[(?, ?, ...)]}
```

Синтаксис хранимой процедуры без параметров:

```
{call имя_процедуры}
```

Создание объекта CallableStatement

```
CallableStatement cstmt = con.prepareCall( "{call
    getTestData(?, ?)}");
```

IN- и OUT-параметры

Передача значений входных парметров объекта CallableStatement осуществляется с помощью методов setXXX, унаследованных от PreparedStatement.

JDBC-типы всех OUT-параметров хранимых процедур должны быть зарегистрирваны перед их вызовом.

Регистрация типов данных выходного параметра производится методом registerOutParameter.

```
CallableStatement cstmt = con.prepareCall( "{call
    getTestData(?, ?)}");
cstmt.registerOutParameter(1,
    java.sql.Types.TINYINT);
cstmt.registerOutParameter(2,
    java.sql.Types.DECIMAL, 3);
cstmt.executeQuery();
byte x = cstmt.getByte(1);
java.math.BigDecimal n = cstmt.getBigDecimal(2, 3);
```

Параметры INOUT

В случае параметра, который одновременно является и входным, и выходным (INOUT), необходимо вызывать как соответствующий метод setXXX (унаследованный от PreparedStatement), так и метод registerOutParameter.

Метод setXXX устанавливает входное значение параметра, а registerOutParameter регистрирует тип выходного значения.

```
CallableStatement cstmt = con.prepareCall( "{call
    reviseTotal(?)}");
cstmt.setByte(1, 25);
cstmt.registerOutParameter(1,
    java.sql.Types.TINYINT);
cstmt.executeUpdate();
byte x = cstmt.getByte(1);
```

Метаданные

- Существует целый ряд методов интерфейсов **ResultSetMetaData** и **DatabaseMetaData** для интроспекции объектов.
- Получить объект **ResultSetMetaData** можно следующим образом:

```
ResultSetMetaData rsMetaData = rs.getMetaData();
```

Считывайте выходные(OUT) параметры после считывания результатов

Рекомендуется сначала считывать результаты, сгенерированные вызовом CallableStatement, а затем выходные (OUT) параметры.

Если объект CallableStatement возвращает несколько объектов ResultSet (с использованием метода execute), то ВСЕ результаты должны быть прочитаны перед первым обращением к выходным параметрам.

После этого значения выходных параметров могут быть извлечены спомощью методов CallableStatement.getXXX.

Методы интерфейса **ResultSetMetaData**

- **int getColumnCount() –** возвращает число столбцов набора результатов объекта **ResultSet;**
- **String getColumnName(int column) –** возвращает имя указанного столбца объекта **ResultSet;**
- **int getColumnType(int column) –** возвращает тип данных указанного столбца объекта **ResultSet**

DatabaseMetaData

```
DatabaseMetaData dbMetaData =
    cn.getMetaData();
```

- String getDatabaseProductName() – возвращает название СУБД;
- String getDatabaseProductVersion() – возвращает номер версии СУБД;
- String getDriverName() – возвращает имя драйвера JDBC;
- String getUserName() – возвращает имя пользователя БД;
- String getURL() – возвращает местонахождение источника данных;
- ResultSet getTables() – возвращает набор типов таблиц, доступных для данной БД