Урок 2

Базы данных

Реляционные базы данных. Язык запросов SQL. Операторы SELECT, INSERT, UPDATE, DELETE. Подключение к базе через JDBC. Отправка запросов и обработка результатов.

Базы данных

Чтобы начать работу с базой данных, нужно установить систему управления (СУБД). Примеры таких систем – MySQL, Oracle, MS SQL, SQLite.

Мы воспользуемся SQLite. Ее особенности:

- хранит всю базу в одном файле;
- не требует установки;
- не поддерживает тип данных Data.

Поддерживаемые типы данных:

- NULL – NULL-значение;
- INTEGER – целое знаковое;
- REAL – с плавающей точкой;
- TEXT – текст, строка (UTF-8);
- BLOB – бинарные данные.

Язык запросов SQL

Аббревиатура CRUD (Create/Read/Update/Delete) обозначает набор операций, которые можно производить над данными в базе. Они выполняются с помощью языка запросов SQL. Все команды языка регистронезависимы, могут быть разделены любым количеством пробелов и переносов строк.

CREATE

```
CREATE TABLE [имя таблицы] (
[имя колонки] [тип данных],
[имя колонки] [тип данных],
... );
```

Пример запроса:

```
CREATE TABLE IF NOT EXISTS Students
(
  StudID INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
  Name TEXT NOT NULL,
  GroupName TEXT NOT NULL,
  Score INTEGER NOT NULL
);
```

Здесь создается таблица **Students** с полями **ID, Name, GroupName, Score**.

NOT NULL означает, что поле всегда должно быть проинициализировано. СУБД следит за этим: если поле равно NULL, выдает ошибку.

PRIMARY KEY указывает, что поле имеет уникальное значение в этой таблице. В нашем примере это поле ID – мы хотим, чтобы у каждой записи был уникальный номер.

AUTOINCREMENT обозначает, что при каждом добавлении записи в таблицу ей автоматически будет присвоен ID на единицу больше предыдущего.

READ

Операция чтения данных из таблицы называется SELECT.

```
SELECT [список полей] FROM [имя таблицы] WHERE [условие];
```

Примеры запросов:

```
SELECT * FROM Students;
SELECT * FROM Students WHERE ID > 3
SELECT GroupName FROM Students WHERE ID = 2
```

Символ «*» означает, что мы хотим получить все поля таблицы. Иначе можно через запятую перечислить необходимые поля. Выражение WHERE не обязательно, но помогает извлекать только интересные для нас данные.

UPDATE

Это операция изменения уже присутствующих в таблице данных или добавления новых.

Добавление новых данных:

```
INSERT INTO [имя таблицы] ([список полей через запятую]) VALUES ([список
значений через запятую]);
```

Изменение:

```
UPDATE [имя таблицы] SET [имя колонки]=[новое значение], [имя колонки]=[новое
значение],.. WHERE [условие];
```

Примеры запросов:

```
INSERT INTO Students (Name, GroupName, Score) VALUES ("Bob",
"Tbz11", 80);
UPDATE Students SET Score = 90 WHERE Name = "Bob";
```

DELETE

Удаление данных из таблицы:

```
DELETE FROM [имя таблицы] WHERE [условие];
```

Пример запроса:

```
DELETE FROM ACCOUNTS WHERE ID='0';
```

JDBC

Каждая СУБД разрабатывается конкретной компанией. Чтобы взаимодействовать с базой данных, производитель выпускает специальный драйвер – JDBC. С его помощью устанавливают соединение, изменяют данные, посылают запросы и обрабатывают их результаты.

Все основные сущности в JDBC API – это интерфейсы: Connection, Statement, PreparedStatement, CallableStatement, ResultSet, Driver, DatabaseMetaData. JDBC-драйвер конкретной базы данных предоставляет их реализации.

DriverManager – это синглтон, который содержит информацию о всех зарегистрированных драйверах. Метод **getConnection** на основании параметра URL находит **java.sql.Driver** соответствующей базы данных и вызывает у него метод **connect**.

Установка соединения

Драйвер JDBC можно скачать с сайта производителя СУБД. Он распространяется в виде .jar – библиотеки, которую необходимо подключить к проекту. Прежде чем использовать драйвер, его нужно зарегистрировать. Имя драйвера можно найти на сайте разработчиков.

```
// Для SQLite регистрация выглядит следующим образом
Class.forName("org.sqlite.JDBC");
// Для H2 Database - org.h2.Driver
// Для MySQL - com.mysql.jdbc.Driver
```

Исходный код реализации любого драйвера будет содержать статический блок инициализации:

```
static {
    try {
        java.sql.DriverManager.registerDriver(new Driver());
    } catch (SQLException e) {
        throw new RuntimeException("Can't register driver!");
    }
}
```

Вызов **Class.forName()** загружает класс и этим гарантирует выполнение статического блока инициализации, а значит и регистрацию драйвера в **DriverManager**. Чтобы указать, как найти базу данных, используется URL – специальная строка формата **[protocol]:[subprotocol]:[name]**:

```
protocol: jdbc
subprotocol: sqlite
name: test.db
Connection conn;
conn = DriverManager.getConnection("jdbc:sqlite:mydatabase.db");
// ... Действия с БД ...
conn.close();
```

Объект **Connection** предоставляет доступ к базе данных. Опционально в него можно передать имя пользователя и пароль, если они установлены. После окончания работы с базой соединение необходимо закрыть методом **close()**.

Запросы в базу

Когда соединение с базой установлено, можно отправлять запросы. Для этого используется объект **Statement**, который умеет хранить SQL-команды. В базу можно отправить запрос на получение или изменение данных. В первом случае результатом будет объект **ResultSet**, который хранит результат. Во втором – количество строк таблицы, которые были изменены.

```
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("SELECT * FROM users");
Statement updateStmt = conn.createStatement();
int result = stmt.executeUpdate("INSERT INTO Students (Name, GroupName, Score)
VALUES ("Bob", "Tbz11", 80);");
```

Подготовленный запрос и пакетное выполнение запросов

Для выполнения множества похожих запросов наиболее эффективным и быстрым решением будет **PreparedStatement** – скомпилированная версия SQL-выражения.

В запросах можно использовать параметры: изменять его динамически в зависимости от входных данных. Параметр заменяется символом «?». Каждому параметру в запросе присваивается порядковый номер – индекс, начиная с 1. У объекта **PreparedStatement** есть методы, которые позволяют установить параметры. Нужно указать их позицию и значение:

```
PreparedStatement ps = conn.prepareStatement("SELECT * FROM Students WHERE ID =
?");
ps.setInt(1, 2);
ResultSet rs = ps.executeQuery();
```

PreparedStatement поддерживает пакетную (batch) отправку SQL-запросов, что значительно уменьшает трафик между клиентом и базой данных:

```
PreparedStatement ps = conn.prepareStatement("INSERT INTO Students(Name,
GroupName) VALUES(?, ?);");
statement.setString(1, "Alex");
statement.setString(2, "Tbz11");
statement.addBatch();
statement.setInt(1, "Sergey);
statement.setString(2, "Tbz11");
statement.addBatch();
statement.executeBatch();
```

Обработка результатов

Результатом запроса **SELECT** в базу является таблица (набор строк), которая сохраняется в объекте **ResultSet**. По строкам можно перемещаться вперед и назад. Для получения значений из определенной колонки текущей строки можно воспользоваться методами **get<Type>(<Param>)**, где **Type** – это тип извлекаемого значения, а **Param** – номер колонки (int) или имя колонки (String).

```
ResultSet rs = stmt.executeQuery();
while (rs.next()) {                    // Пока есть строки
  String name = rs.getString(2);       // Или rs.getString("Name");
}
rs.first();                            // Перейти к первой строке
rs.last();                             // Перейти к последней
rs.next();                             // Перейти к следующей
rs.previous();                         // Перейти к предыдущей
```

Закрытие ресурсов

На каждое соединение СУБД выделяет определенные ресурсы, количество которых ограничено. Поэтому после окончания работы с объектами соединения их нужно закрывать.

Транзакции в JDBC

По умолчанию каждое SQL-выражение автоматически коммитится при выполнении **statement.execute()** и подобных методов. Чтобы открыть транзакцию, сначала необходимо установить флаг **autoCommit** у соединения в значение **false**, а затем пользоваться методами **commit()** и **rollback()**.

```
conn.setAutoCommit(false);
Statement st = conn.createStatement();
try {
    st.execute("INSERT INTO user(name) values('kesha')");
    conn.commit();
} catch (SQLException e) {
    conn.rollback();
}
```

Домашнее задание

1. Сформировать таблицу товаров (id, prodid, title, cost) запросом из Java-приложения:
    - id – порядковый номер записи, первичный ключ;
    - prodid – уникальный номер товара;
    - title – название товара;
    - cost – стоимость.
2. При запуске приложения очистить таблицу и заполнить 10000 товаров вида:
    - id_товара 1 товар1 10
    - id_товара 2 товар2 20
    - id_товара 3 товар3 30
    - …
    - id_товара 10000 товар10000 100000
3. Написать консольное приложение, которое позволяет узнать цену товара по его имени, либо вывести сообщение «Такого товара нет», если товар не обнаружен в базе. Консольная команда: «/цена товар545».
4. Добавить возможность изменения цены товара. Указываем имя товара и новую цену. Консольная команда: «/сменитьцену товар10 10000».
5. Вывести товары в заданном ценовом диапазоне. Консольная команда: «/товарыпоцене 100 600».

Дополнительные материалы

1. Кей С. Хорстманн, Гари Корнелл. Java. Библиотека профессионала. Том 1. Основы;
2. Стив Макконнелл. Совершенный код;
3. Брюс Эккель. Философия Java;
4. Герберт Шилдт. Java 8: Полное руководство;
5. Герберт Шилдт. Java 8: Руководство для начинающих.

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. Герберт Шилдт. Java. Полное руководство // 9-е изд.: Пер. с англ. – М.: Вильямс, 2012. – 1 376 с.