# Базы данных

План урока

- виды баз данных
- язык запросов SQL
- оператор select
- CRUD
- JDBC - работа с базой из Java
- Создание и удаление таблиц

## Виды баз данных

База данных - совокупность данных, организованных в соответствие со структурой, описывающей характеристики этих данных и взаимоотношения между ними.

- иерархические
- реляционные
- объектно-ориентированные

Реляционные базы данных хранят всю информацию в двумерных таблицах, связанных между собой специальными отношениями. Использую эти отношения, можно извлекать информацию из разных таблиц с помощью одного запроса.

В курсе Java 2 нас будут интересовать именно реляционные базы данных. Чтобы начать работу с базой данных, нужно установить у себя систему управления базой данных (СУБД). Примеры таких систем - MySQL, Oracle, MS SQL. Мы будем использовать легковесную СУБД SQLite (http://www.sqlite.org/download.html скачивать Precompiled Binaries for Windows)

Она не требует установки и хранит все данные в одном файле. Имеет интерфейс командной строки (http://www.sqlite.org/cli.html) Несколько примеров использования (Приложение называется sqlite3.exe)

```
sqlite3 my_database.db // создаст базу данных с именем
   my_database.db

C:\javacourse>sqlite3 mydb.db
SQLite version 3.8.4.3 2014-04-03 16:53:12
Enter ".help" for usage hints.
sqlite> create table users (
   ...> name text,
   ...> password text);
sqlite> .tables
users
sqlite> insert into users (name, password) values ("Bob",
   "123");
sqlite> select * from users;
Bob|123
sqlite>
```

Полный список команд можно прочитать, набрав .help внутри консоли SQLite.

## Особенности SQLite

- хранит всю базу в одном файле
- не требует установки
- не требует авторизации (мы не создаем пользователя и пароль)
- не поддерживает тип данных Дата

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 содержит в себе все операции, которые можно производить над данными в базе. CRUD = Create/Read/Update/Delete

Эти операции выполняются с помощью специального языка - SQL. Все команды должны заканчиваться символом точка с запятой (;). Все команды языка регистронезависимы и могут иметь между собой любое количество пробелов и переносов строк.

**CREATE**

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

```
CREATE TABLE IF NOT EXISTS ACCOUNTS
(
   ID       INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
   DESCR    TEXT                      NOT NULL,
   USER_NAME TEXT                     NOT NULL
);
```

Здесь создается таблица Account с полями ID, DESCR, USERNAME.

NOT NULL - означает, что поле всегда должно быть проинициализировано. СУБД следит за этим и в случае, если поле равно NULL выкидывает ошибку.

PRIMARY KEY - означает, что поле имеет уникальное значение в этой таблице (В примере это поле ID - мы хотим, чтобы у каждой записи был уникальный номер).

AUTOINCREMENT - означает, 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];

```
SELECT * FROM ACCOUNTS
WHERE ID='3';
// ID | DESRC | USER_NAME
// 3   RUB    BOB
```

```
SELECT USER_NAME FROM ACCOUNTS;
// USER_NAME
//  Bob
// Alisa
```

Символ звездочка означает, что мы хотим получить все поля таблицы. Иначе мы можем через запятую перечислить необходимые поля.

Выражение WHERE необязательно, но помогает извлекать только интересные для нас данные.

**UPDATE**

Операция изменения уже присутствующих в таблице данных либо добавления новых.

Добавление новых данных:

INSERT INTO [имя таблицы] ([список полей через запятую])

VALUES ([список значение через запятую]);

Изменение:

UPDATE [имя таблицы]

SET [имя колонки]=[новое значение], [имя колонки]=[новое значение],..

WHERE [условие];

```
INSERT INTO ACCOUNTS (DESCR, USER_NAME) VALUES ("RUB",
   "John");

UPDATE ACCOUNTS SET USER_NAME="Ivan"
WHERE USER_NAME="John";
```

**DELETE**

Удаление данных из таблицы.

DELETE FROM [имя таблицы] WHERE [условие];

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

Подробно описание всех операций SQL можете найти на http://www.w3schools.com/sql

# JDBC

Каждая СУБД разрабатывается отдельной компанией. Чтобы можно было работать с базой, производитель пишет специальный драйвер базы - jdbc.

Java с помощью этого драйвера может обращаться к базе данных и взаимодействовать с ней. JDBC имеет следующие возможности:

- устанавливать соединение с базой данных
- посылать запросы и изменять данные (CRUD)
- обрабатывать результаты запросов

## Установка соединения

Драйвер jdbc обычно можно скачать с сайта производителя СУБД. Он распространяется в виде .jar - библиотеки. Эту библиотеку нужно подключить к вашему проекту (добавить в classpath). Прежде чем использовать драйвер, его нужно зарегистрировать - имя драйвера можно также найти на сайте разработчиков.

Чтобы указать, как найти базу данных используется URL - специальная строка формата [protocol]:[subprotocol]:[name]
protocol: jdbc
subprotocol: sqlite
name: mydatabase.db

---

```
import java.sql.Connection;
import java.sql.DriverManager;

Connection conn;

void connect() {
      Class.forName("org.sqlite.JDBC");
      String databaseUrl = "jdbc:sqlite:mydatabase.db";
      conn = DriverManager.getConnection(databaseUrl);
}

void close() {
```

```
        conn.close();
}
```

Объект Connection предоставляет доступ к базе данных. Опционально в объект Connection можно передать пользователя и пароль(если используете). После окончания работы с базой, соединение нужно закрыть методом close().

## Запросы в базу

После того, как соединение с базой установлено, можно отправлять запросы. Для этого используем объект Statement, который умеет хранить SQL команды. В базу можно отправить запрос на получение данных, либо на изменение. В первом случае результатом будет объект ResultSet, который хранит результат. Во втором случае - 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 users
   (username, login) VALUES ("Bob Jr", "bob.junior");");
```

## Подготовленный запрос

В запросах вы можете использовать параметры, то есть изменять его динамически в зависимости от входных данных. Параметр заменяется символом ?. Каждому параметру в запросе присваивается порядковый номер - индекс. Индексы начинаются с 1. У объекта PreparedStatement есть методы, которые позволяют установить параметры, вам нужно указать позицию и значение параметра.

```
PreparedStatement pStmt = conn.prepareStatement("SELECT *
   FROM users WHERE login = ?");
pStmt.setString(1, "bob.junior");
ResultSet rs = pStmt.executeQuery();
```

## Обработка результатов

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

```
ResultSet rs = stmt.executeQuery();
while (rs.next()) { // пока есть строки
      String name = rs.getString(2);
      // String name = rs.getString("username");
}

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 и количество их ограничено, поэтому после окончания работы с объектами соединения их нужно закрывать.

```
// old variant jvm <=1.6

Statement stmt = null;
ResultSet rs = null;
try {
      stmt = conn.createStatement();
      rs = stmt.executeQuery("your query");
      // process data
} catch (SQLException e) {
      logger.error(e.getMessage());
} finally {

      // Эти процедуры лучше вынести в отдельные
```

```
        // статические утилитные методы
        // closeResource(stmt);
        try {
              if (rs != null)
                    rs.close();
        } catch (SQLException e) {
              logger.warn("Failed to close ");
        }

        try {
              if (stmt != null)
                    stmt.close();
        } catch (SQLException e) {
              logger.warn("Failed to close");
        }
}


// jvm >=1.7 try-with-resources

// ресурс автоматически будет закрыт после выхода из блока
try (Statement stmt = conn.createStatement()) {
        ...

} catch (SQLException e) {
        ...
}
```

---

Объект соединения можно закрыть в конце работы с базой.

# Материалы к уроку

- Хорстманн К., Корнелл Г. Java 2. Том 2. Тонкости программирования (7-е издание, 2007), глава 4 - работа с базами данных
- http://docs.oracle.com/javase/tutorial/jdbc/basics/
- SQL шпаргалка http://www.w3schools.com/sql
- SQLite http://www.sqlite.org/

- SQLite jdbc https://bitbucket.org/xerial/sqlite-jdbc/downloads

# Домашнее задание

1. установить себе СУБД SQLite (можете использовать любую другую по своему выбору), скачать драйвер jdbc для нее.
2. научиться пользоваться интерфейсом командной строки СУБД, попробовать создать базу, таблицу. Попробуйте настроить работу с базой из вашей IDE.
3. выполнить скрипты create.sql, insert.sql (приложены), чтобы создать тестовую базу.
4. написать следующие запросы в базу данных:
    - вывести имена (username) тех пользователей, на которых подписан aruizca;
    - вывести количество зарегистрированных пользователей;
    - вывести все сообщения от тех пользователей, на которых подписан aruizca, (*) отсортировать сообщения по дате. На выходе нужны поля сообщение, дата;
    - (*) вывести количество подписчиков для каждого пользователя, отсортировать по убыванию. На выходе поля логин пользователя, количество подписчиков;

## Задача финансовый менеджер

Продолжаем заниматься приложением менеджер финансов. На данный момент у нас имеется набор классов Пользователь, Счет, Транзакция. Есть интерфейс DataStore (смотрите прошлый урок). В приложение можно залогиниться с помощью пары логин/пароль. Теперь реализуем возможность сохранять данные в базу, чтобы при перезапуске они не потерялись. Дополнительно введем сущность Категория - на что были потрачены деньги (Здоровье, Еда, Одежда, Путешествие и т.д.). Каждая транзакция должна иметь поле Категория.

1. реализовать класс Category - у категории есть описание, которое можно изменять, категории можно добавлять и удалять. Добавить поле категория в класс транзакции.

2. реализовать класс DbHelper, который умеет подключаться к базе и возвращать объект Connection:

   - если в базе еще нет таблиц (первый запуск приложения), то таблицы должны быть созданы
   - придумайте, как реализовать DbHelper, чтобы его можно было создать только один раз (см. шаблон проектирования Singletone)
   - у класса есть один публичный метод

```
public class DbHelper {
      public Connection getConn() {
            ...
   }
}
```

3. спроектировать базу данных для финансового менеджера. Подумайте, какие таблицы вам нужны, как вы будете определять уникальность записи (первичный ключ).

4. реализовать интерфейс DataStore с помощью базы данных

5. (*) вместо того, чтобы хранить пароль в открытом виде, храните только его хэш. Хэш вычисляется, когда пользователь создает аккаунт, затем сохраняется в базе (оригинальный пароль не сохраняется нигде). При авторизации сравнивайте хэш введенного пароля с хэшом из базы данных. Java: класс MessageDigest, Wiki - хэширование.

6. при вводе пароля с консоли, он не должен печататься, либо заменять звездочками.

7. ваши идеи для этого приложения - делитесь!