Лабораторная работа 4

Работа с БД

I. Теоретическая часть

Практически одновременно с выпуском первой реализации JDK была опубликована и спецификация JDBC.

JDBC – это API для выполнения SQL-запросов к базам данных. Представляет набор классов и интерфейсов, определенных в пакете `java.sql`.

JDBC предполагает передачу запроса к базе данных в виде строки. Таким образом, могут использоваться конструкции SQL, специфичные для данной базы данных и/или ее JDBC-драйвера.

Определяя универсальный метод работы с базами данных, JDBC предполагает и конкретные требования к базовому SQL. Базовым требованием JDBC является удовлетворение входного уровня (*Entry Level*) ANSI-стандарта SQL-92.

Увеличение числа методов интерфейсов, с одновременной упрощением функциональности этих методов. Определяя работу по принципу "одно действие – один метод", этот подход должен обеспечить лучшую читаемость и прозрачность логики кода.

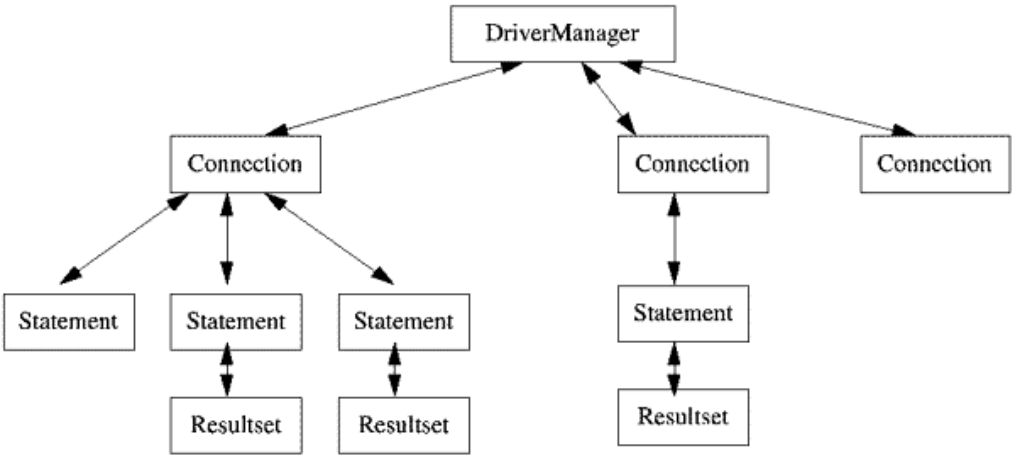

Рисунок 1 – Схема взаимодействия интерфейсов JDBC

`java.sql.DriverManager` обеспечивает загрузку драйверов и создание новых соединений (`connection`) с базой данных; это интерфейс JDBC, определяющий корректный выбор и инициализацию драйвера для данной СУБД в данных условиях;

`java.sql.Connection` определяет характеристики и состояние соединения с БД; кроме того, он предоставляет средства для контроля транзакций и уровня их изолированности;

`java.sql.Statement` выполняет функции контейнера по отношению к SQL-выражению; при этом под выражением понимается не только сам текст запроса, но и такие характеристики, как параметры и состояние выражения;

`java.sql.ResultSet` предоставляет доступ к набору строк, полученному в результате выполнения данного SQL-выражения.

Интерфейс выражения `java.sql.Statement` выступает в качестве предка для других двух важных интерфейсов: `java.sql.PreparedStatement` и `java.sql.CallableStatement`, первый из которых предназначен для выполнения прекомпилированных SQL-выражений, второй - для выполнения вызовов хранимых процедур. Соответственно `Statement` выполняет обычные (*статические*) SQL-запросы, а указанные два наследника работают с параметризированными SQL-выражениями.

II. Практическая часть.

1. Создание базы данных и таблиц

Создавать БД, таблицы в ней, а также выполнять манипуляции с хранящимися данными можно двумя способами: в самой программе или вспомогательными инструментальными средствами.

Для MySQL используется инструмент **MySQL Workbench**. Он позволяет выполнять задачи администрирования, работы с таблицами и данными, просмотр статистики и многое другое.

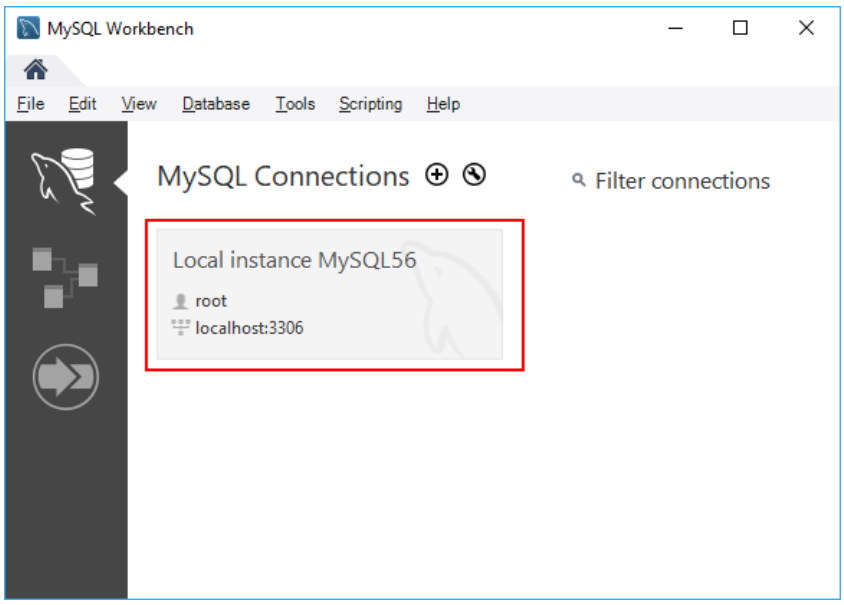

Рисунок 2 – Графический клиент MySQL Workbench для работы с сервером

В левой части в окне SCHEMAS можно увидеть доступные базы данных.

Теперь посмотрим, как мы можем выполнять в этой программе запросы к БД. Вначале создадим саму БД. Для этого нажмем над списком баз данных на значок "SQL" с плюсом:

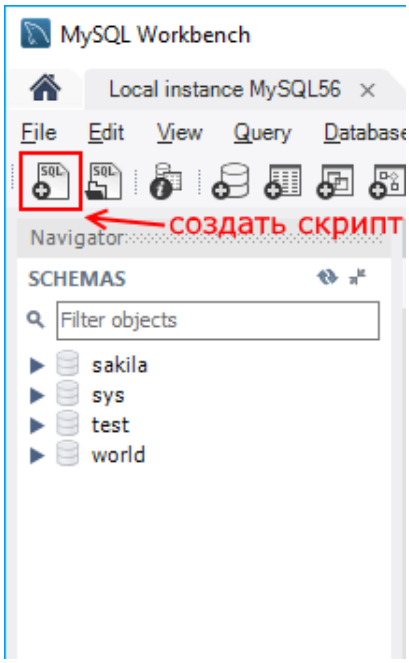

Рисунок 3 – Создание скрипта SQL

После этого в центральной части программы откроется окно для ввода скрипта SQL. Для выполнения скрипта в панели инструментов необходимо нажать на значок молнии.

Создадим БД *institut*, добавим в нее таблицу *Students* с полями *ID*, *Age*, *FirstName*, *LastName*. Внесем в нее несколько строк.

```
create database institut;
use institut;
create table Students (
   Id int auto_increment primary key,
   Age int,
   FirstName varchar(20),
   LastName varchar(20)
);
insert Students(Age, FirstName, LastName) values (20, 'Иван', 'Петров');
insert Students(Age, FirstName, LastName) values (19, 'Маша', 'Иванова');
```

2. Организация соединения с базой данных

JDBC предусматривает возможность динамического подсоединения к базе данных. В этом случае необходима явная загрузка драйвера

```
Class.forName("com.mysql.jdbc.Driver");
```

Драйвер можно скачать вручную и положить в папку с проектом, либо добавить соответствующую зависимость для maven-проекта.

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.33</version>
</dependency>
```

Драйверы JDBC должны реализовывать интерфейс драйвера, а реализация должна содержать статический инициализатор, который будет вызываться когда драйвер загружен. Этот инициализатор регистрирует новый экземпляр сам по себе с DriverManager.

Файл с драйвером необходимо подключить к проекту как библиотеку.

Также нужно указать в файле module-info.java строку

```
requires java.sql;
```

И когда вы вызываете `Class.forName(String className)`, согласно документации API, происходит следующее: вызов `forName("X")` вызывает инициализацию класса с именем `X`.

Метод forName выбрасывает проверяемое исключение `ClassNotFoundException`. Поэтому вызов метода необходимо осуществлять в `try…catch`, например:

```
try {
    Class.forName("com.mysql.cj.jdbc.Driver");
    System.out.println("Driver loading succesfull.");
} catch (ClassNotFoundException ex) {
    System.out.println(ex.getMessage());
    System.out.println("Driver loading failed!");
}
```

Замечение: Методы `DriverManager getConnection` и `getDrivers` были расширены. Драйверы JDBC 4.0 должны включать файл META-INF/услуги/java.sql.Driver. Этот файл содержит имя Драйвер JDBC для java.sql.Driver, это позволяет обойтись без явной загрузки драйвера.

Для доступа к базе данных необходимо получить объект *java.sql.Connection*. Сделать это можно, обратившись к уровню управления JDBC посредством вызова *java.sql.DriverManager.getConnection*. Основным параметром этого вызова является строка URL (*Uniform Resource Locator*).

Механизм именования баз данных в JDBC, базируясь на URL, решает такие вопросы, как автоматический выбор драйвера, способного осуществить доступ к данной БД, определение характеристик соединения и т.п.

В общем случае спецификация рекомендует следующую конструкцию

```
jdbc:<имя драйвера>:[//хост[: порт/]]<имя БД>
```

Пример:

```
final static String DB_URL =
          "jdbc:mysql://localhost:3306/institut";
…
Connection connection =
     DriverManager.getConnection(DB_URL, LOGIN, PASSWORD)
```

Благодаря использованию конструкции *try-with-resources* ресурсы будут освобождены после того, как они больше не нужны. Это убережёт от "провисших" соединений и утечек памяти:

```
try (Connection connection =
```

```
   DriverManager.getConnection(DB_URL, LOGIN, PASSWORD)) {
    System.out.println("Connection succesfull.");
    //код для работы с соединением
} catch (SQLException ex) {
    System.out.println(ex.getMessage());
    System.out.println("Connection error!");
}
```

Для взаимодействия с базой данных приложение отправляет серверу MySQL команды на языке SQL. Чтобы выполнить команду, вначале необходимо создать объект `Statement`.

Для его создания у объекта `Connection` вызывается метод `createStatement()`:

```
Statement statement = connection.createStatement();
```

Создавать таблицы, добавлять и удалять данные можно программно с помощью метода

```
int executeUpdate("Команда_SQL")
```

Например:

```
String sqlCommand = " CREETE TABLE Students (
   Id INT AUTO_INCREMENT PRIMARY KEY,
   Age INT,
   FirstName VARCHAR(20),
   LastName VARCHAR(20)
);
```

В данной работе таблица уже была создана в MySQL Workbench, поэтому создавать ее в коде нет необходимости.

Для выборки данных с помощью команды *SELECT* применяется метод `executeQuery()`:

```
ResultSet resultSet = statement
                    .executeQuery("SELECT Age, FirstName,
                                   LastName FROM Students");
```

Метод возвращает объект `ResultSet`, который содержит все полученные данные.

В объекте `ResultSet` итератор устанавливается на позиции перед первой строкой. И чтобы переместиться к первой строке (и ко всем последующим)

необходимо вызвать метод `next()`. Пока в наборе `ResultSet` есть доступные строки, метод `next()` будет возвращать `true`.

```
while (resultSet.next()){
     String name = resultSet.getString("FirstName");
     int age = resultSet.getInt("Age");
     System.out.printf("Name: %s, age: %d%n", name, age);
}
```

После перехода к строке мы можем получить ее содержимое. Для этого у `ResultSet` определен ряд методов. Некоторые из них:

`getBoolean()` возвращает значение `boolean`

`getDate()` возвращает значение `Date`

`getDouble()` возвращает значение `double`

`getInt()` возвращает значение `int`

`getFloat()` возвращает значение `float`

`getLong()` возвращает значение `long`

`getNString()` возвращает значение `String`

`getString()` возвращает значение `String`

В зависимости от того, данные какого тип хранятся в том или ином столбце, мы можем использовать тот или иной метод. Каждый из этих методов имеет две версии:

```
int getInt(int columnIndex)
int getInt(String columnLabel)
```

Первая версия получает данные из столбца с номером *columnIndex*. Вторая версия получает данные из столбца с названием *columnLabel*.

Добавление данных

Для добавления данных в БД применяется команда *INSERT*. Выполнить ее можно также вызовом метода *executeQuery()*.

```
int rows = statement
     .executeUpdate(
          "INSERT Students(Age, FirstName, LastName)
          VALUES (33, 'Густав', 'Шнайдер');
```

Метод возвращает значение 1 если данные были добавлены, и 0, если нет.

CRUD-операции

При работе с базами данных наиболее часто используемые операции это *добавление*, *изменение* и *удаление* данных. В SQL этим функциям, операциям соответствуют операторы *Insert* (создание записей), *Select* (чтение записей), *Update* (редактирование записей), *Delete* (удаление записей).

Добавление

```
Statement statement = connection.createStatement();
int rows = statement.
    executeUpdate("INSERT Students(Age, FirstName, LastName)
                  VALUES (19, 'Петр', 'Иванов'),
                  (18, 'Маша', 'Петрова'),
                  (21, 'Петр', 'Козловский')");
    System.out.printf("Добавлено %d записей", rows);
```

Диалоговые окна

Класс `Alert` является подклассом `Dialog` и предоставляет поддержку для некоторых видов уведомлений.

`Alert` по умолчанию является окном с модальностью (modality) `Modelity.WINDOW_MODAL`. Можно изменить используя метод `alert.initModality(Modality)`.

Существует 3 вида, которые можно применить:

- `Modality.NONE`
- `Modality.WINDOW_MODAL`
- `Modality.APPLICATION_MODAL`

`Modality.NONE` – когда открывается новое окно с этой модальностью, новое окно будет независимым по отношению к родительскому окну.

`Modality.WINDOW_MODAL` – новое окно блокирует родительское окно.

`Modality.APPLICATION_MODAL` – окно блокирует все другие окна приложения. Вы не можете взаимодействовать ни с каким окном, до тех пор пока это окно не закроется.

Диалоговое окно содержит следующие области, задаваемые программно:

Header Region – заголовок окна.

Content Region – заголовок и содержание уведомления.

Footer Region – отображает кнопки в зависимости от типа кона.

По умолчанию *Content Region* отображает `Label`, но можно настроить текстовое содержание данному через метод `alert.setContentText(String)`. Так же (при необходимости) можно отобразить другой Node в *Content Region* через `alert.getDialogPane().setContent(Node)`.

```
Alert alert = new Alert(AlertType.INFORMATION);
alert.setTitle("Информация");
alert.setHeaderText("Результат выполнения операции");
alert.setContentText("Запрошенная операция выполнена успешно. Данные обновлены. ");
alert.showAndWait();
```

Если в дополнительном заголовке нет необходимости, то можно не задавать данный параметр или указать на его отсутствие в явном виде:

```
alert.setHeaderText(null);
```

В качестве типа можно использовать следующие:

```
AlertType.WARNING
AlertType.INFORMATION
AlertType.ERROR
```

Задание на лабораторную работу.

Задание 1.

1. Установить и настроить СУБД MySQL
2. Установить средство MySQL Workbench
3. Создать базу данных INSTITUT
4. Добавит в БД таблицу STUDENTS, содержащую поля имя, фамилию, отчество, группа, возраст, город.
5. Добавить три записи, используя MySQL Workbench.

Написать программу:

1. Подключиться к СУБД
2. По нажатию на кнопку вывести на форму в компонент `TextField` или `TextArea` содержимое таблицы STUDENTS.
3. Получить с полей на форме новые значения для записи в таблицу.
4. Добавить в таблицу введенные значения.
5. Убедиться, что данные добавлены. Информировать пользователя используя диалоговое окно.

Задание 2.

Создать и заполнить БД в соответствии с вариантом.

Вариант 1	*Геометрические фигуры*	Вариант 7	*Учет аудиторного фонда*
Вариант 2	*Библиотека*	Вариант 8	*Учет оборудования*
Вариант 3	*Магазин продуктов*	Вариант 9	*Экзаменационная комиссия*
Вариант 4	*Магазин промтоваров*	Вариант 10	*Дистанционное управление техникой*
Вариант 5	*Отдел кадров*	Вариант 11	*Планировка помещений*
Вариант 6	*Автомастерская*	Вариант 12	*Интернет-провайдер*

Реализовать:

- добавление в БД записей, введенных пользователем;
- отбор и вывод всех записей, содержащие данные, введенные пользователем на форме. Для отбора записей использовать одно из полей БД.
- удаление записей по заданному условию.

Информировать о результатах операций (число добавленных, отобранных или удаленных записей) с использованием диалогового окна.