

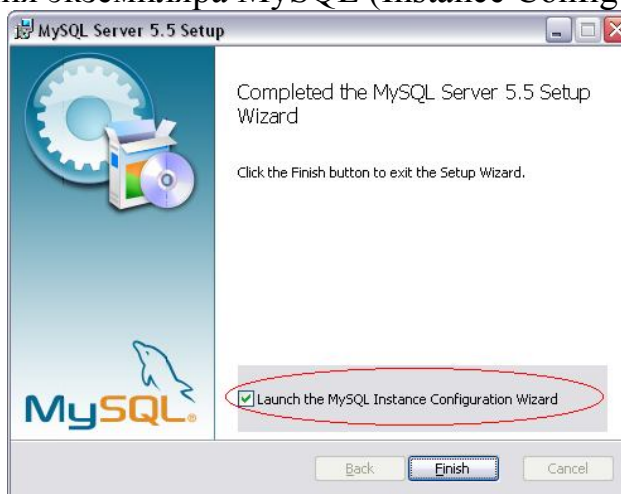
2. Рекомендации по выполнению работы

2.1. Установка и конфигурирование СУБД

Для выполнения лабораторной работы потребуется установить на компьютер и сконфигурировать СУБД.

В данной лабораторной работе рекомендуется использовать свободную СУБД MySQL, распространяемую по лицензии [GNU General Public License](http://www.gnu.org/licenses/gpl.html). Дистрибутив СУБД MySQL доступен на официальном сайте MySQL по ссылке: <http://www.mysql.com/downloads/>.

Для установки MySQL запустите инсталлятор (файл вида mysql.xxxx.msi) и пройдите все шаги мастера установки, оставив предлагаемые по умолчанию значения параметров установки. По завершению установки MySQL запустите мастер конфигурирования экземпляра MySQL (Instance Configuration Wizard):



В мастере конфигурирования также используйте значения, предлагаемые по умолчанию.

Обратите внимание на параметры связи с конфигурируемым экземпляром. Запомните порт TCP (3306 по умолчанию) и используйте его в дальнейшем при подключении к СУБД из приложения.



На одном из шагов мастера потребуется ввести пароль (с подтверждением) администратора экземпляра (root). Запомните введенный пароль и используйте его в дальнейшем при работе с СУБД.



2.2. Создание базы данных

В установленной СУБД создадим новую базу данных с таблицами для хранения объектов в соответствии с заданной предметной областью. Например, для варианта №1 потребуется создать две таблицы: таблицу стран и таблицу городов.

Администрирование СУБД MySQL можно осуществлять через консольное приложение MySQL Command Line Client:



Для создания новой базы данных используется команда CREATE DATABASE.

В следующем примере мы создадим новую базу данных с названием MAP:
CREATE DATABASE MAP;



Перед созданием таблиц необходимо подключиться к определенной базе данных. Для этих целей в MySQL используется команду USE. Команда USE предписывает MySQL использовать указанную базу данных по умолчанию в последующих запросах.

USE MAP;



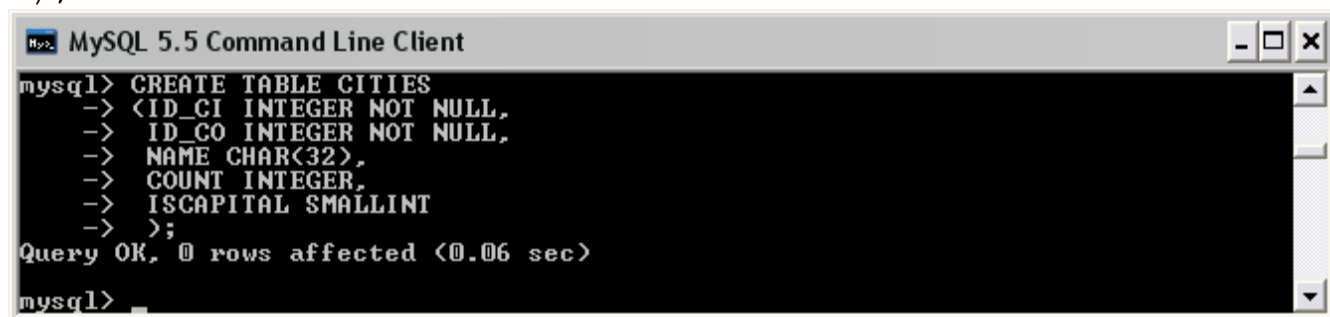
Создание таблиц осуществляется при помощи запроса CREATE TABLE, например:

```
CREATE TABLE COUNTRIES  
(ID_CO INTEGER NOT NULL,  
NAME CHAR(32));
```



```
MySQL 5.5 Command Line Client  
mysql> CREATE TABLE COUNTRIES  
-> <ID_CO INTEGER NOT NULL,  
-> NAME CHAR(32)>;  
Query OK, 0 rows affected (0.05 sec)  
mysql>
```

```
CREATE TABLE CITIES  
(ID_CI INTEGER NOT NULL,  
ID_CO INTEGER NOT NULL,  
NAME CHAR(32),  
COUNT INTEGER,  
ISCAPITAL SMALLINT  
);
```



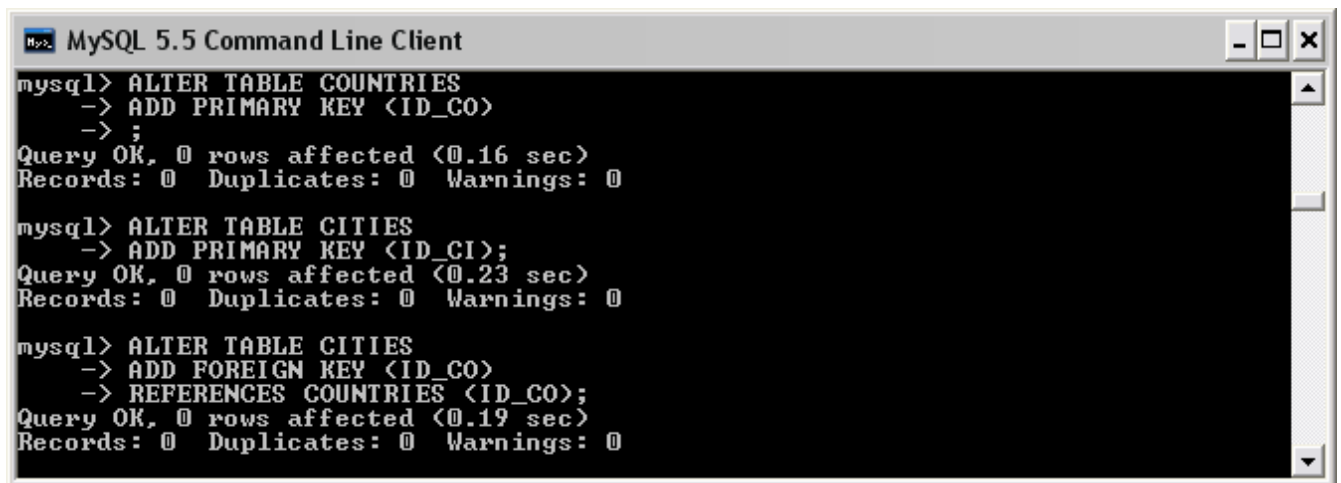
```
MySQL 5.5 Command Line Client  
mysql> CREATE TABLE CITIES  
-> <ID_CI INTEGER NOT NULL,  
-> ID_CO INTEGER NOT NULL,  
-> NAME CHAR(32),  
-> COUNT INTEGER,  
-> ISCAPITAL SMALLINT  
-> >;  
Query OK, 0 rows affected (0.06 sec)  
mysql>
```

Для обеспечения уникальности идентификаторов, а также целостностей связей между сущностями, следует использовать первичные и внешние ключи:

```
ALTER TABLE COUNTRIES  
ADD PRIMARY KEY (ID_CO);
```

```
ALTER TABLE CITIES  
ADD PRIMARY KEY (ID_CI);
```

```
ALTER TABLE CITIES  
ADD FOREIGN KEY (ID_CO)  
REFERENCES COUNTRIES (ID_CO);
```



```
mysql> ALTER TABLE COUNTRIES
-> ADD PRIMARY KEY <ID_CO>
-> ;
Query OK, 0 rows affected (0.16 sec)
Records: 0 Duplicates: 0 Warnings: 0

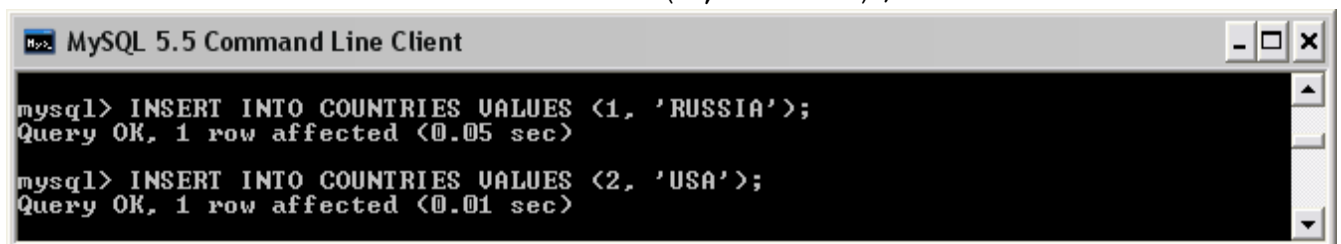
mysql> ALTER TABLE CITIES
-> ADD PRIMARY KEY <ID_CI>;
Query OK, 0 rows affected (0.23 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> ALTER TABLE CITIES
-> ADD FOREIGN KEY <ID_CO>
-> REFERENCES COUNTRIES <ID_CO>;
Query OK, 0 rows affected (0.19 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

2.3. Работа с данными

Заполнить таблицы данными можно при помощи SQL-оператора INSERT, например:

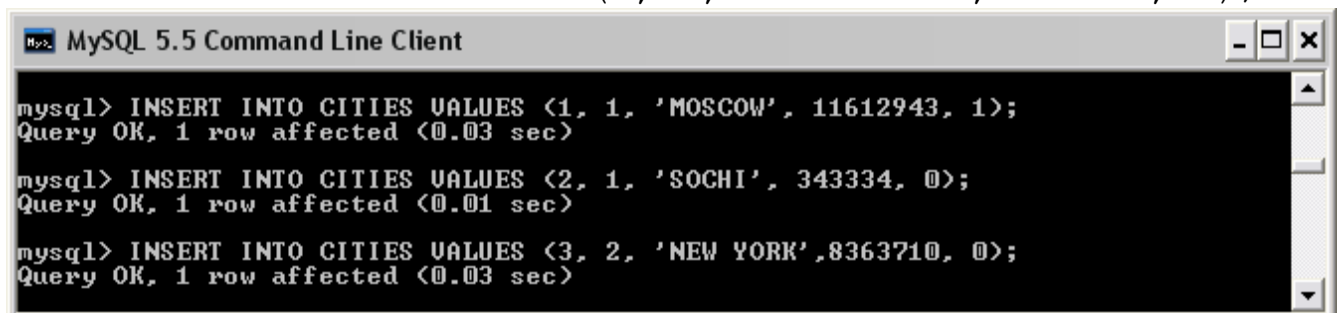
```
INSERT INTO COUNTRIES VALUES (1, 'RUSSIA');
INSERT INTO COUNTRIES VALUES (2, 'USA');
```



```
mysql> INSERT INTO COUNTRIES VALUES (1, 'RUSSIA');
Query OK, 1 row affected (0.05 sec)

mysql> INSERT INTO COUNTRIES VALUES (2, 'USA');
Query OK, 1 row affected (0.01 sec)
```

```
INSERT INTO CITIES VALUES (1, 1, 'MOSCOW', 11612943, 1);
INSERT INTO CITIES VALUES (2, 1, 'SOCHI', 343334, 0);
INSERT INTO CITIES VALUES (3, 2, 'NEW YORK', 8363710, 0);
```



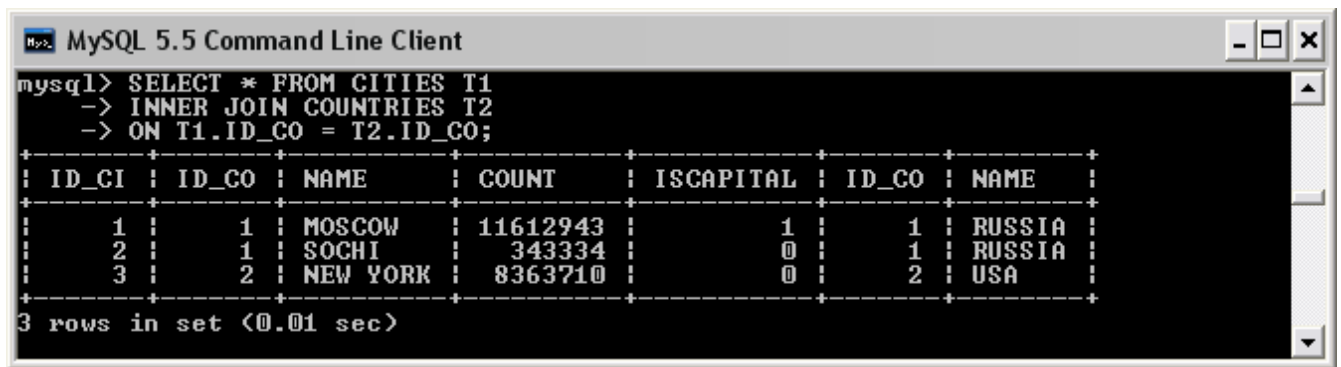
```
mysql> INSERT INTO CITIES VALUES (1, 1, 'MOSCOW', 11612943, 1);
Query OK, 1 row affected (0.03 sec)

mysql> INSERT INTO CITIES VALUES (2, 1, 'SOCHI', 343334, 0);
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO CITIES VALUES (3, 2, 'NEW YORK', 8363710, 0);
Query OK, 1 row affected (0.03 sec)
```

Выборка данных из таблиц осуществляется при помощи оператора SELECT, например:

```
SELECT * FROM CITIES T1
INNER JOIN COUNTRIES T2
ON T1.ID_CO = T2.ID_CO;
```

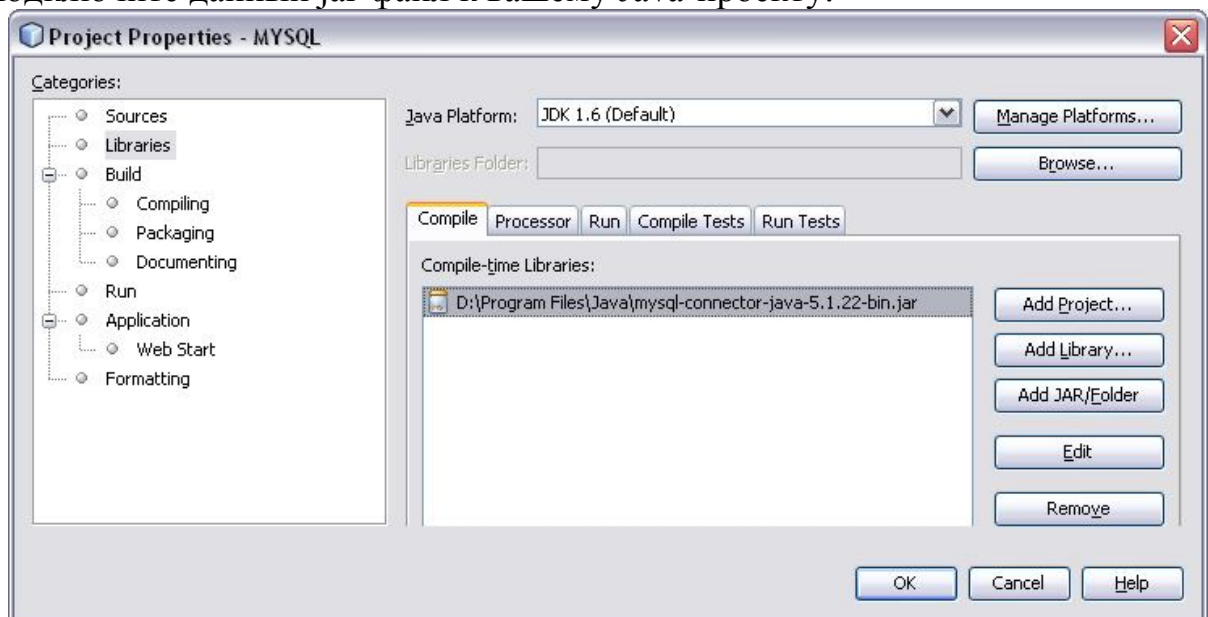


2.4. Доступ к MySQL из приложения Java

Для доступа к базе данных MySQL через интерфейс JDBC потребуется библиотека *mysql-connector-java*. Данная библиотека доступна на официальном сайте MySQL (<http://www.mysql.com/downloads/connector/j/>).

Загрузите данную библиотеку, распакуйте архив и найдите в нем jar-файл вида *mysql-connector-java.xx.xx.xx-bin.jar*. Данный jar-файл содержит драйвер *com.mysql.jdbc.Driver*, обеспечивающий работу с СУБД MySQL через интерфейс JDBC.

Подключите данный jar-файл к вашему Java-проекту:



Подключите к проекту библиотеку *java.sql.**, содержащую классы JDBC:

```
import java.sql.*;
```

В начале работы вашего Java-приложения загрузите драйвер *com.mysql.jdbc.Driver* через *Class.forName*:

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

2.5. Установка соединения с базой данных из приложения Java

Установка соединения с базой данных должна выполняться в начале работы программы. Полученное соединение будет использоваться в процессе работы программы при выполнении операций над данными.

Управление соединением осуществляется через класс *java.sql.Connection*. Для установки соединения используется метод *getConnection* класса *java.sql.DriverManager*. Метод принимает на вход параметры соединения:

- url, определяющий протокол и идентификатор базы данных (для JDBC типа 4 указывается сетевое имя сервера, порт, имя БД)
- имя пользователя (для удобства можно работать под администратором: *root*)
- пароль

и в случае успеха возвращает объект класса *Connection*.

В случае возникновения ошибки метод *getConnection* генерирует исключение *SQLException*.

В следующем примере мы установим соединение с базой данных *map*, размещающейся на локальном сервере (localhost, порт 3306), используя имя пользователя *root* и пароль *password*.

```
// подключаю драйвер
Class.forName("com.mysql.jdbc.Driver").newInstance();
// устанавливаю соединение с БД
String url = "jdbc:mysql://localhost:3306/map";
Connection con =
    DriverManager.getConnection(url, "root", "password");
// работаю с данными
...
// завершаю соединение
con.close();
```

2.6. Выполнение запросов из приложения Java

Выполнение запросов SQL осуществляется через класс *java.sql.Statement*.

Объект класса *Statement* создается в рамках заданного соединения при помощи метода *createStatement* класса *Connection*:

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

Изменение данных осуществляется через метод *executeUpdate* класса *Statement*. Метод принимает на вход строку с запросом SQL типа INSERT, UPDATE или DELETE и возвращает количество измененных записей в таблице.

```
String sql = "INSERT INTO COUNTRIES VALUES (3, 'CHINA')";
s.executeUpdate(sql);
```

Выборка данных осуществляется при помощи метода *executeQuery*. Метод принимает на вход SQL-запрос вида SELECT и возвращает объект класса *ResultSet*, позволяющий обрабатывать результирующие таблицы выборок.

Класс *ResultSet* содержит следующие основные методы:

- next – переходит к новой записи в выборке (перед работой с выборкой необходимо один раз вызвать данный метод для перехода к первой записи);
- getXXXX (getInt, getString, и т.п.) – методы возвращают значение заданного столбца выборки для текущей записи выборки;
- close – завершает работу с выборкой.

В следующем примере мы выберем из таблицы COUNTRIES все записи и выведем их в консоль:

```
Statement s = con.createStatement();
String sql = "SELECT * FROM COUNTRIES";
ResultSet rs = s.executeQuery(sql);
while (rs.next())
{
    int id      = rs.getInt("ID_CO");
    String name = rs.getString("NAME");
    System.out.println(id + " " + name);
}
rs.close();
```

В случае возникновения ошибки методы класса *Statement* генерируют исключение *SQLException*.

2.7. Пример программы JDBC

Следующая программа реализует добавление, удаление и запрос данных о странах, хранящихся в таблице COUNTRIES базы данных MAP. Каждая страна характеризуется уникальным идентификатором и названием.

```
import java.sql.*;
public class Map
{
    private Connection con = null; // соединение с БД
    private Statement stmt = null; // оператор

    // Конструктор
    public Map(String DBName, String ip, int port)
        throws Exception
    {
        Class.forName("com.mysql.jdbc.Driver").newInstance();
        String url = "jdbc:mysql://" + ip + ":" + port + "/" + DBName;
        con = DriverManager.getConnection(url, "root", "root");
        stmt = con.createStatement();
    }
}
```

```

// Завершение работы
public void stop() throws SQLException
{
    con.close();
}

// Добавление страны
public boolean addCountry(int id, String name)
{
    String sql = "INSERT INTO COUNTRIES (ID_CO, NAME)" +
        "VALUES (" + id + ", '" + name + "')";

    try
    {
        stmt.executeUpdate(sql);
        System.out.println("Страна " + name +
            " успешно добавлена!");
        return true;
    } catch (SQLException e)
    {
        System.out.println("ОШИБКА! Страна " + name +
            " не добавлена!");
        System.out.println(">> " + e.getMessage());
        return false;
    }
}

// Удаление страны
public boolean deleteCountry(int id) throws SQLException
{
    String sql = "DELETE FROM COUNTRIES WHERE ID_CO = " + id;
    try
    {
        int c = stmt.executeUpdate(sql);
        if (c > 0)
        {
            System.out.println("Страна с идентификатором "
                + id + " успешно удалена!");
            return true;
        }
        else
        {
            System.out.println("Страна с идентификатором "
                + id + " не найдена!");
            return false;
        }
    } catch (SQLException e)
    {
        System.out.println(
            "ОШИБКА при удалении страны с идентификатором " + id);
        System.out.println(">> " + e.getMessage());
        return false;
    }
}

```



```

// Запрос всех стран
public void showCountries()
{
    String sql = "SELECT ID_CO, NAME FROM COUNTRIES";
    try
    {
        ResultSet rs = stmt.executeQuery(sql);
        System.out.println("СПИСОК СТРАН:");
        while (rs.next())
        {
            int id      = rs.getInt("ID_CO");
            String name = rs.getString("NAME");
            System.out.println(" >> " + id + " - " + name);
        }
        rs.close();
    } catch (SQLException e)
    {
        System.out.println(
            "ОШИБКА при получении списка стран");
        System.out.println(" >> " + e.getMessage());
    }
}

// ТЕСТОВЫЙ СЦЕНАРИЙ
public static void main(String[] args) throws Exception
{
    Map m = new Map("map", "localhost", 3306);
    m.showCountries();
    m.addCountry(1, "RUSSIA");
    m.addCountry(5, "JAPAN");
    m.addCountry(6, "UKRAINE");
    m.deleteCountry(3);
    m.deleteCountry(7);
    m.showCountries();
    m.stop();
}
}

```

Возможный результат работы программы:

СПИСОК СТРАН:

```

>> 1 - RUSSIA
>> 2 - USA
>> 3 - CHINA

```

ОШИБКА! Страна RUSSIA не добавлена!

```
>> Duplicate entry '1' for key 'PRIMARY'
```

Страна JAPAN успешно добавлена!

Страна UKRAINE успешно добавлена!

Страна с идентификатором 3 успешно удалена!

Страна с идентификатором 7 не найдена!

СПИСОК СТРАН:

```

>> 1 - RUSSIA
>> 2 - USA
>> 5 - JAPAN
>> 6 - UKRAINE

```