**Основная цель туторила – научиться обращаться к табличным данным посредством jdbc/odbc драйвера**

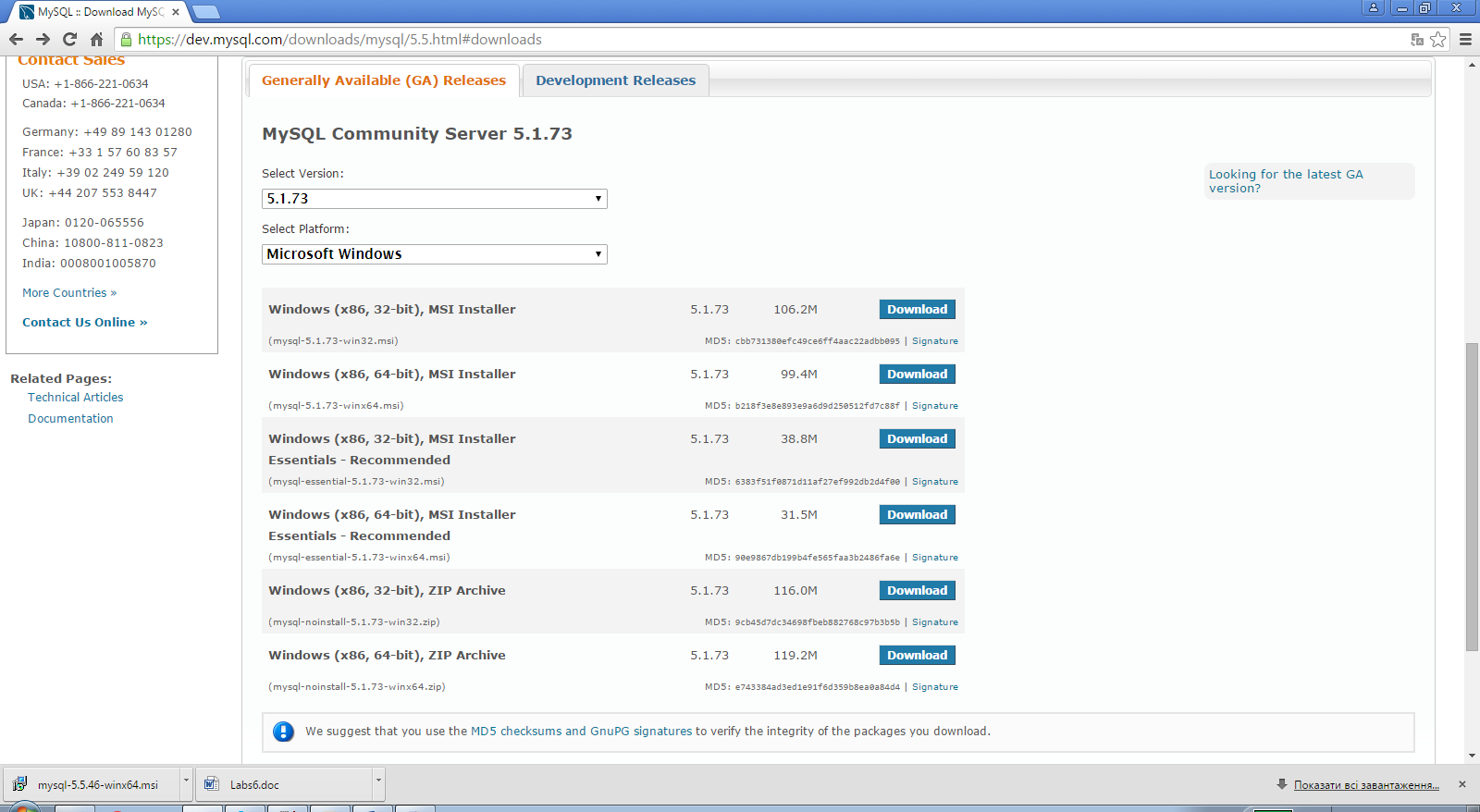
Для этого рассмотрим 2 случая:

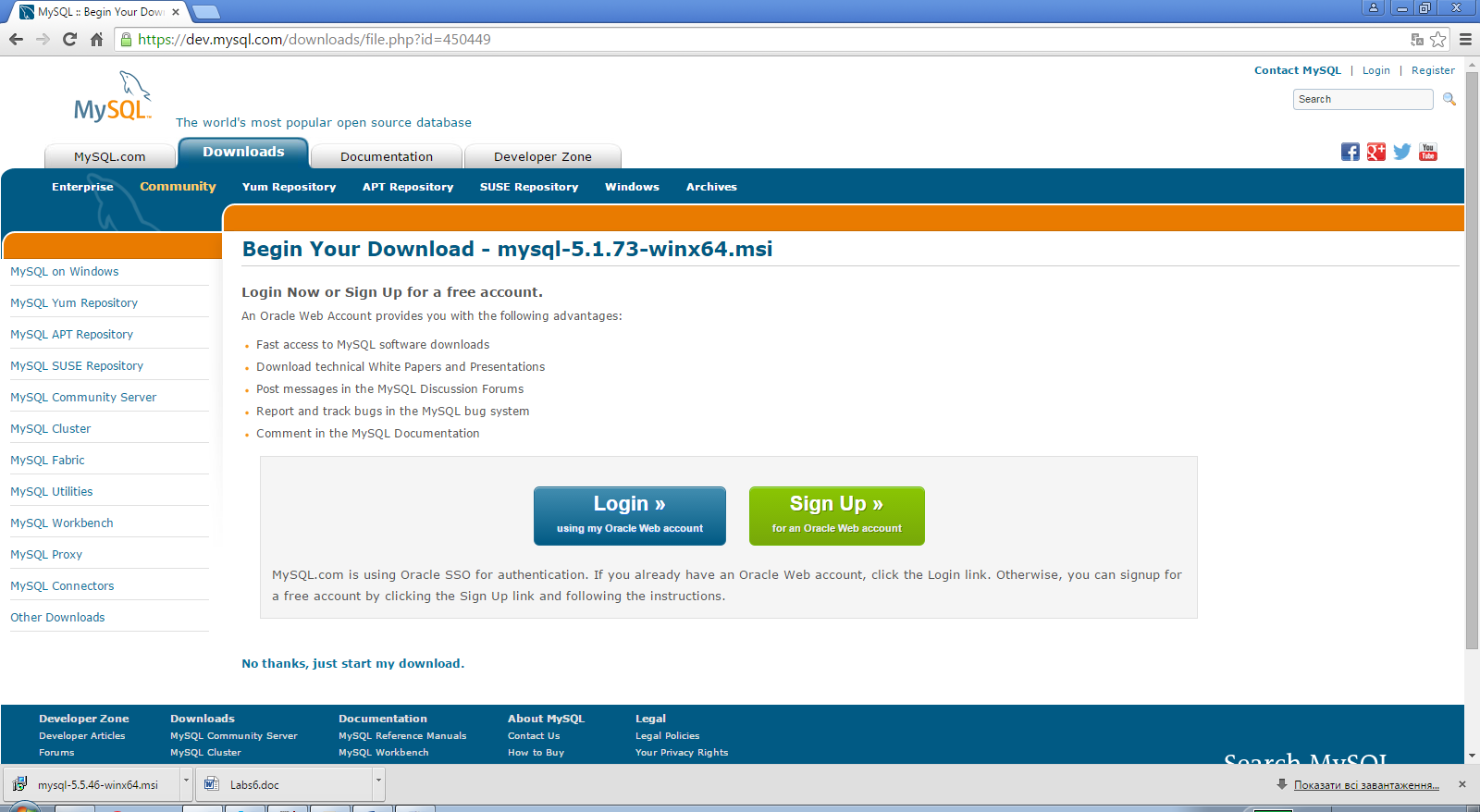
1. Использование «чистого» jdbc драйвера;
2. Использование odbc;

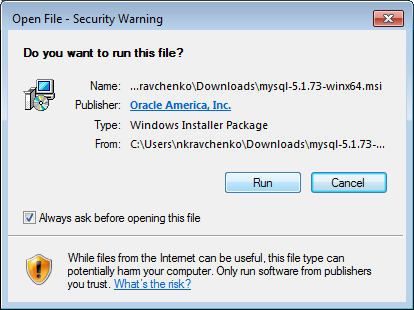
Показывать буду на примере связки c MySQL:

1. **Для начала установим MySQL сервер**

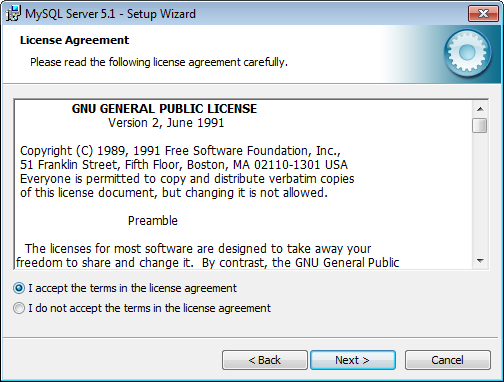
<https://dev.mysql.com/downloads/mysql/5.5.html#downloads>

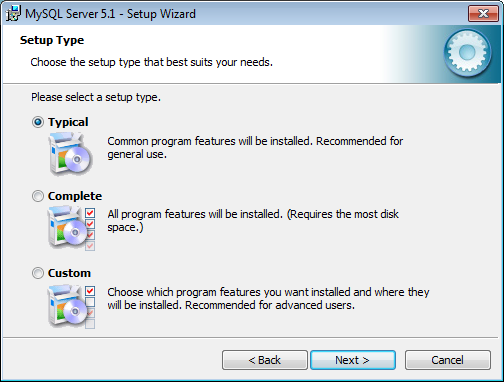


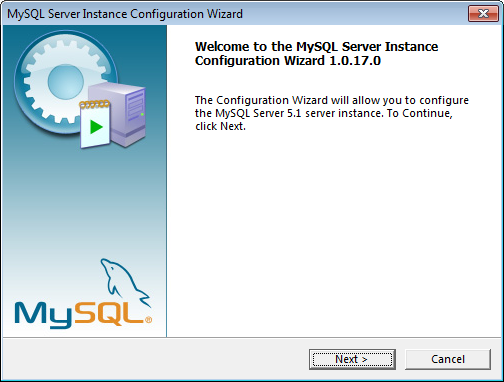


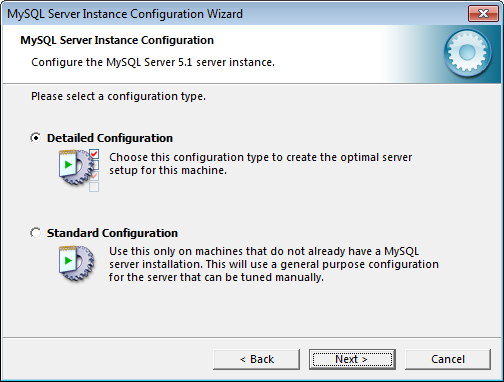


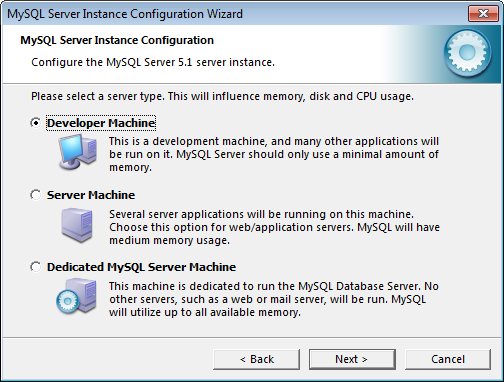


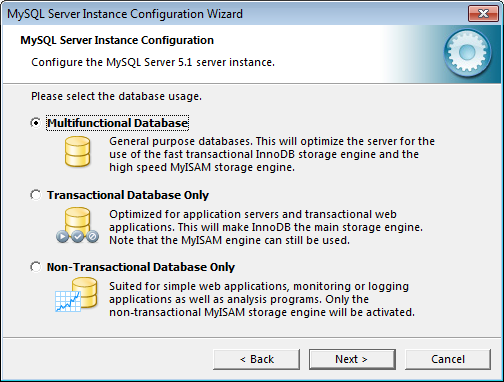


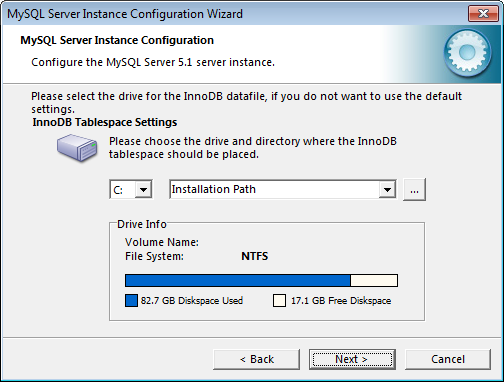


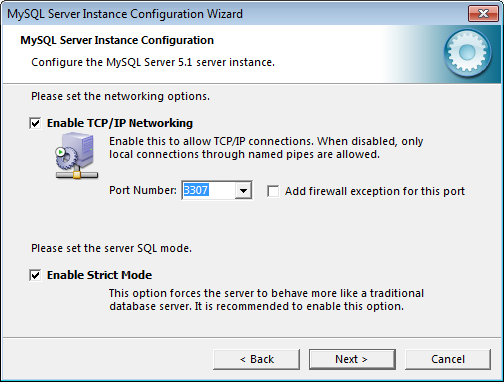


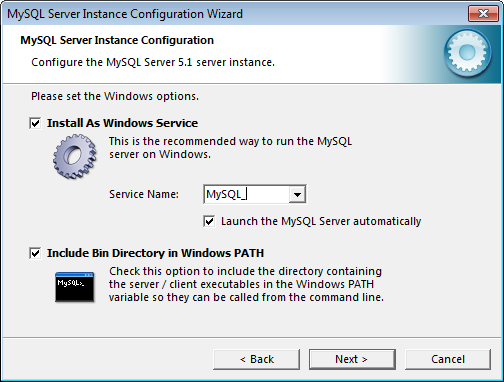


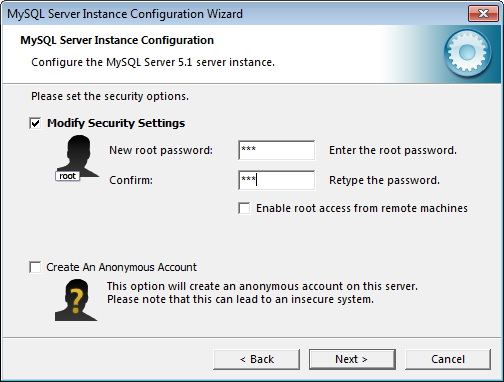


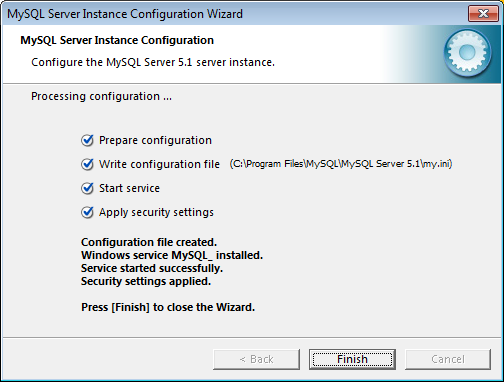




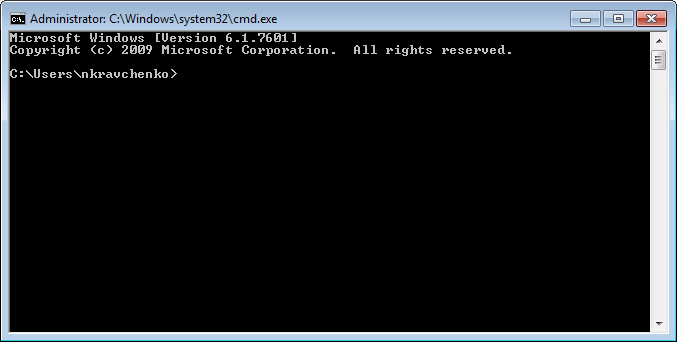




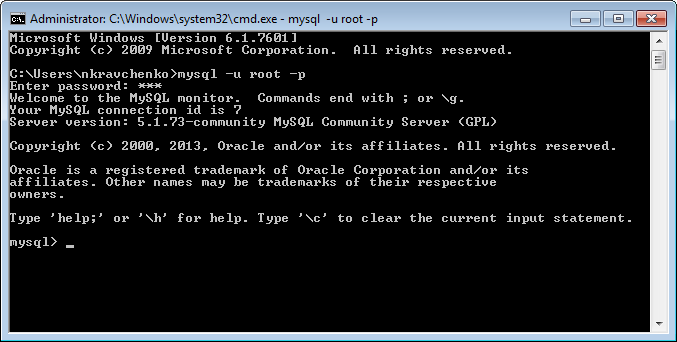




1. **Откроем console:**

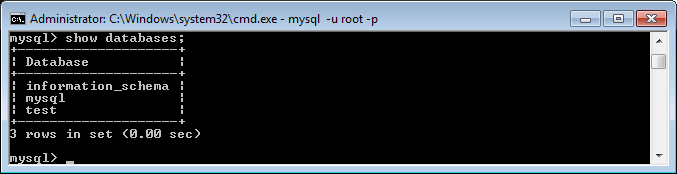


Введем **mysql –u root –p** затем **Enter**



Посмотрим базы, которые изначально существуют в системе, для этого выполним запрос вида:

**show all databases;**



Создадим тестовую базу – например **test\_jdbc\_db:**

**create database test\_jdbc\_db;**

Для того, чтобы переключиться на ее использование – выполним:

**use test\_jdbc\_db;**

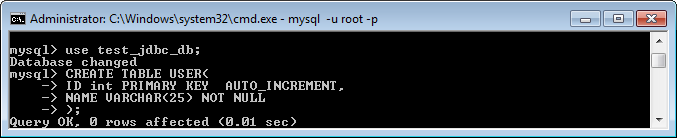
1. Создадим простую табличку(Вы в будущем можете развернуть нормальную схему DB)

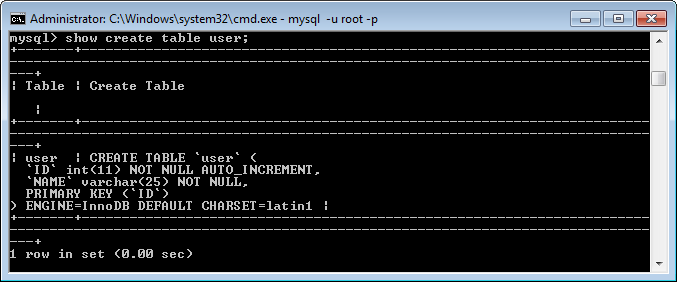
**CREATE TABLE USER(**

**ID int PRIMARY KEY AUTO\_INCREMENT,**

**NAME VARCHAR(25) NOT NULL**

**);**

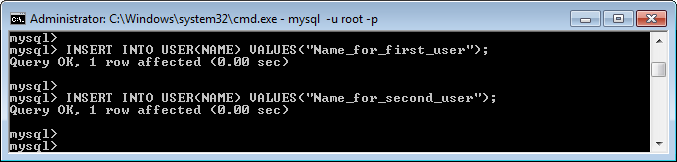




1. Добавим несколько записей в табличку:

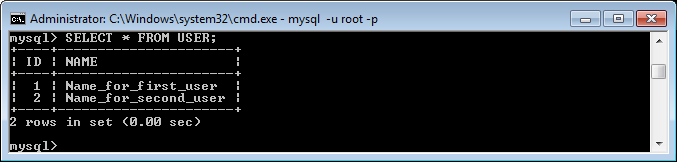
**INSERT INTO USER(NAME) VALUES(“Name\_for\_first\_user”);**

**INSERT INTO USER(NAME) VALUES(“Name\_for\_second\_user”);**



1. Сделаем выборку из таблички, проверим, что все гуд:

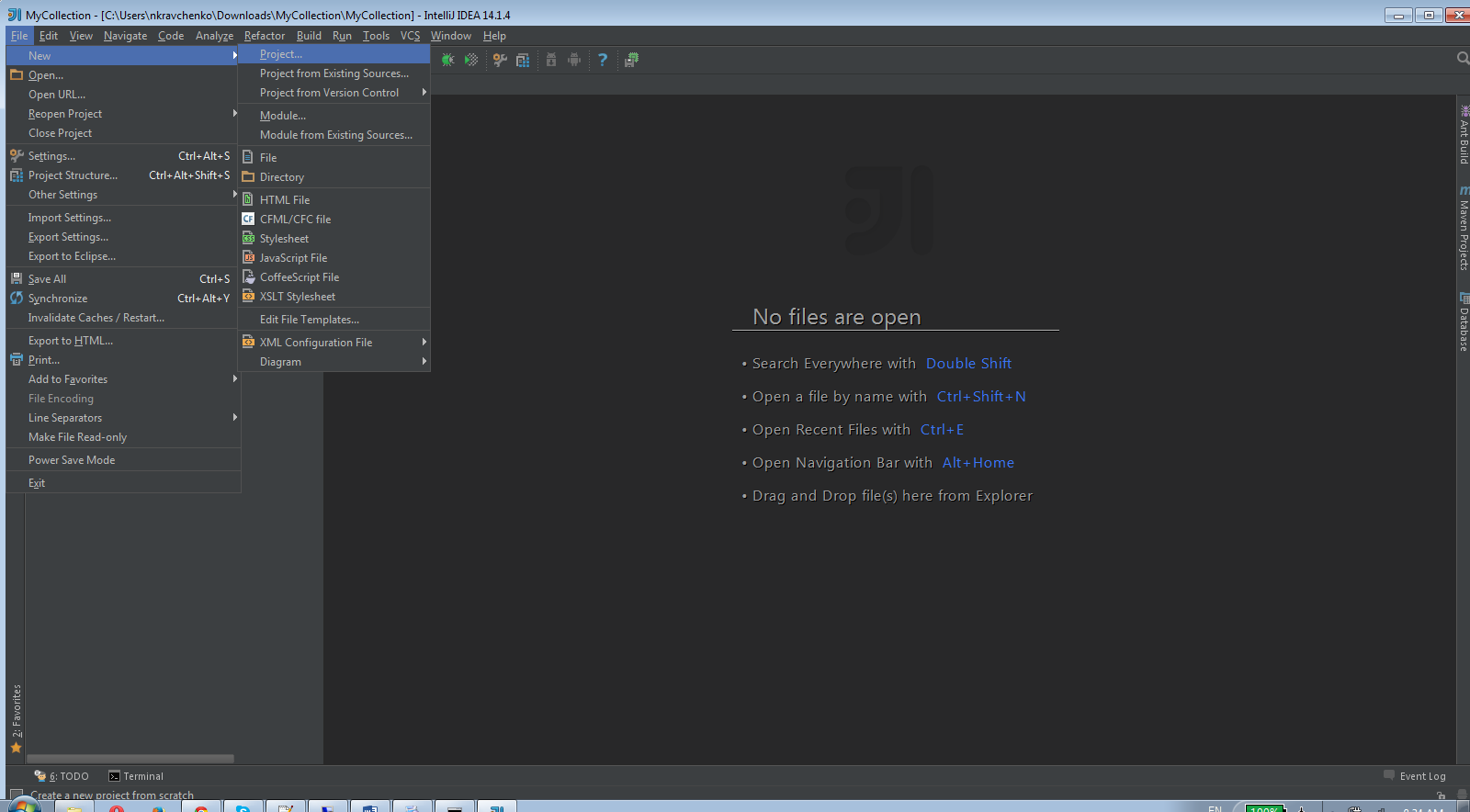
**SELECT \* FROM USER;**



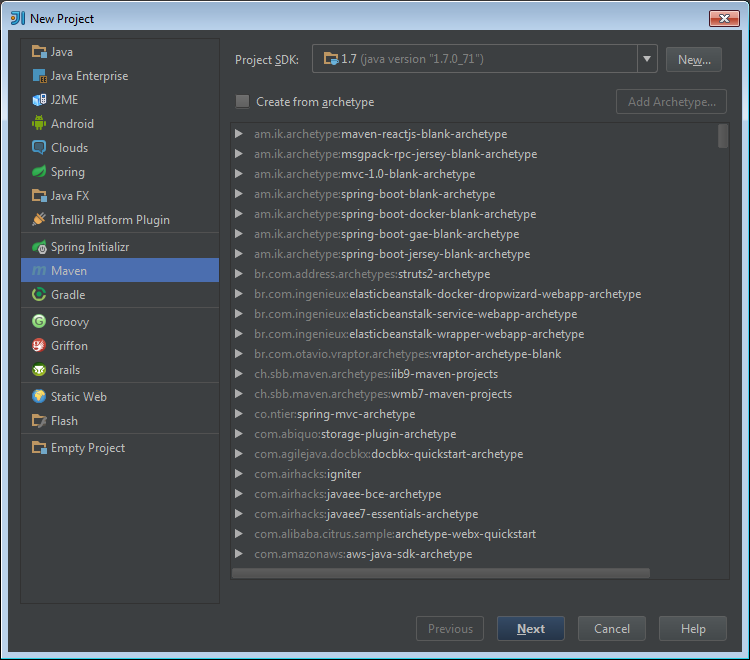
1. Перейдем к коду

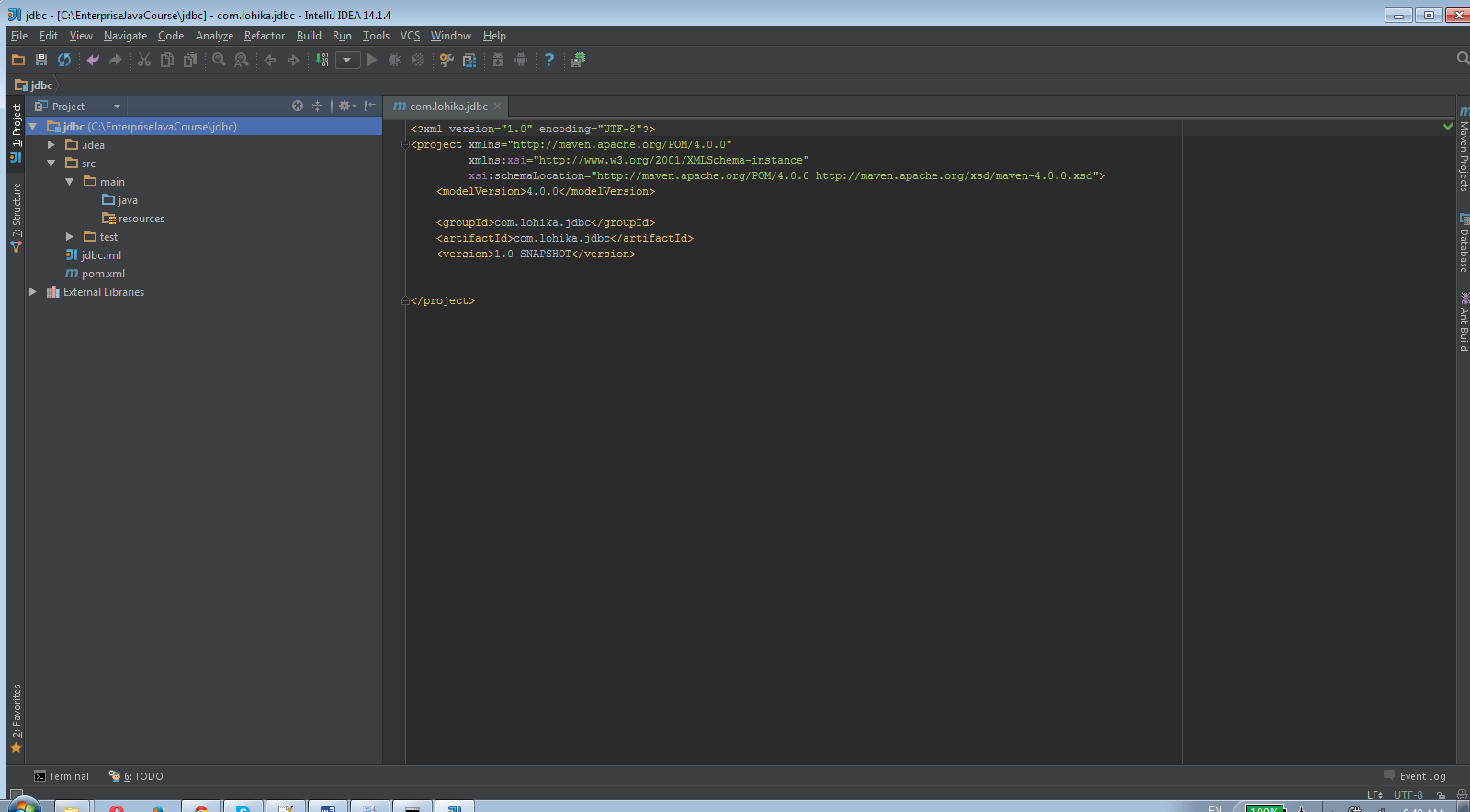
Запустим **IDE**



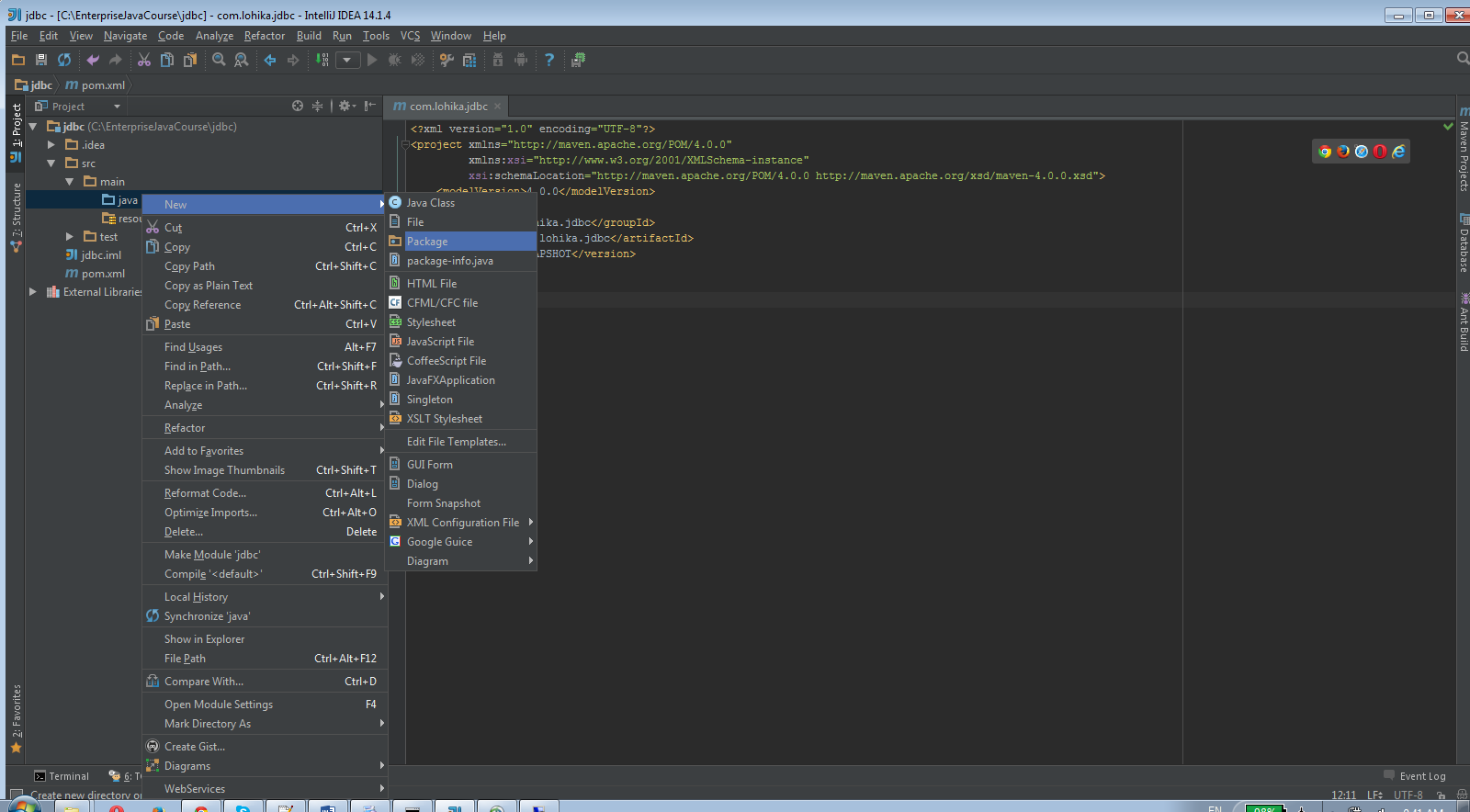


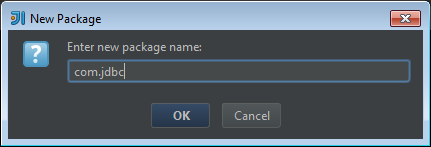
Создадим проект (**Maven**)



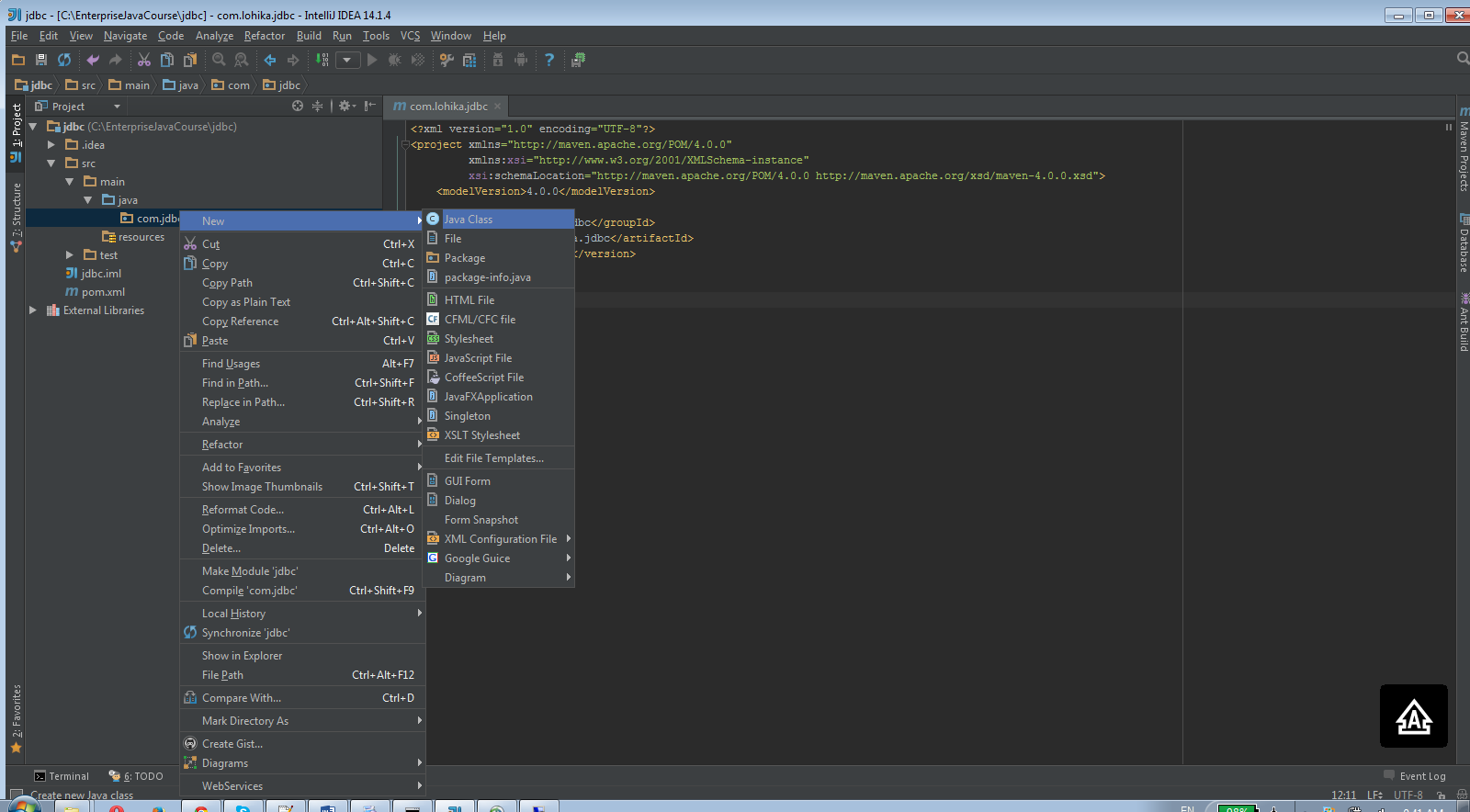


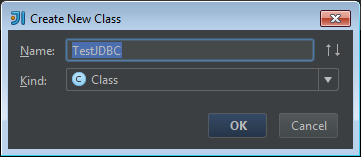
Cоздадим пакет **com.jdbc**





Создадим класс TestJDBC:





1. Добавим в **pom.xml** нужный депенденси – для того, чтобы мы смогли подключиться к источнику данных, указанного выше провайдера (**MySQL**)

<http://mvnrepository.com/artifact/mysql/mysql-connector-java/5.1.6>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.6</version>

</dependency>

**pom.xml**

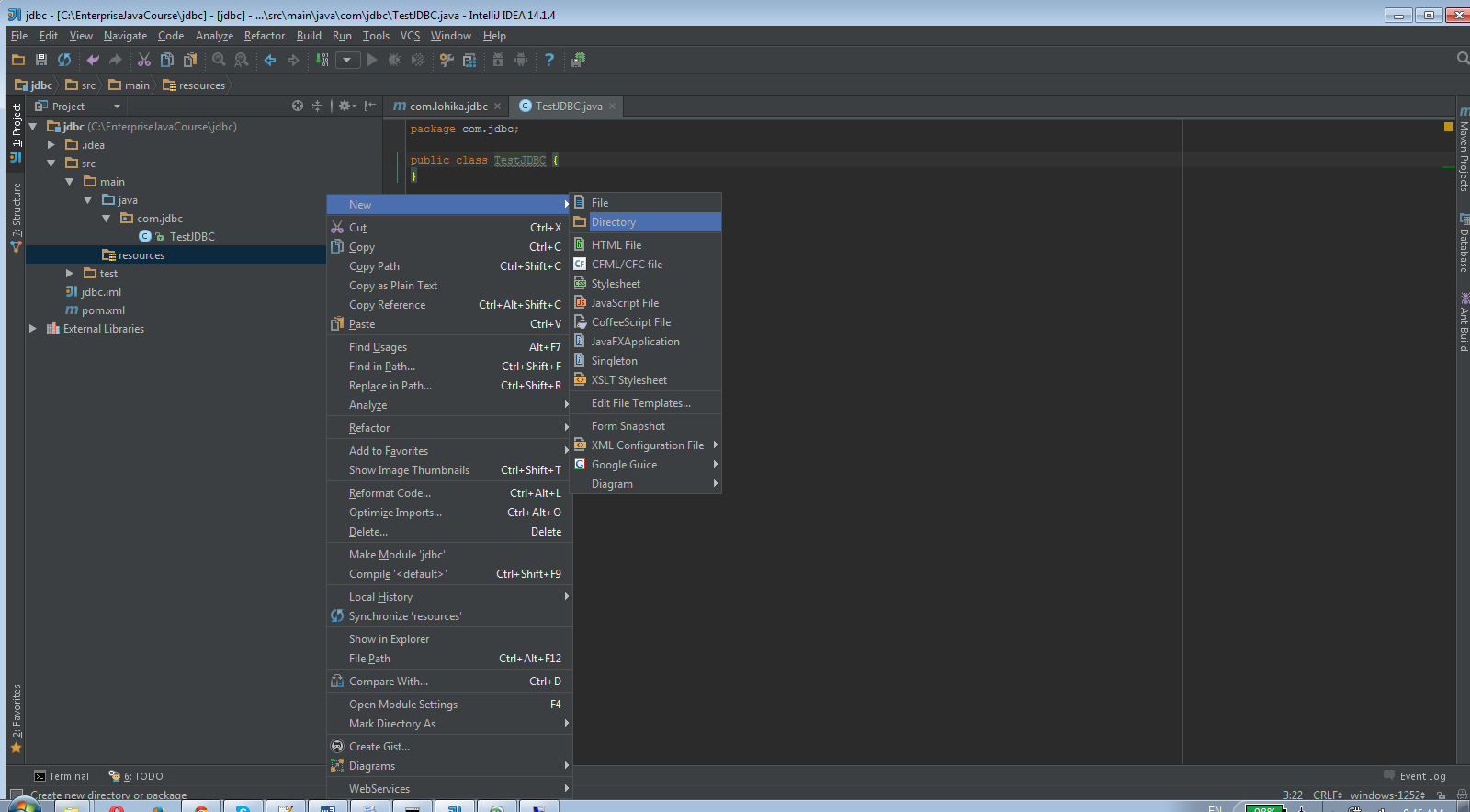
<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>com.test.jdbc</groupId>  
 <artifactId>com.test.jdbc</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 <dependencies>  
 <dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
 <version>5.1.6</version>  
 </dependency>  
 </dependencies>  
  
</project>

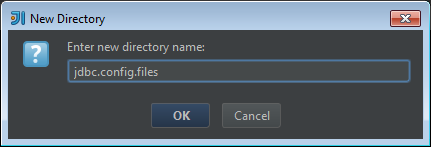
1. Предварительно создадим property file, содержащий necessary jdbc-parameters (**username, password, url** для коннекта)

**В нашем случае:**

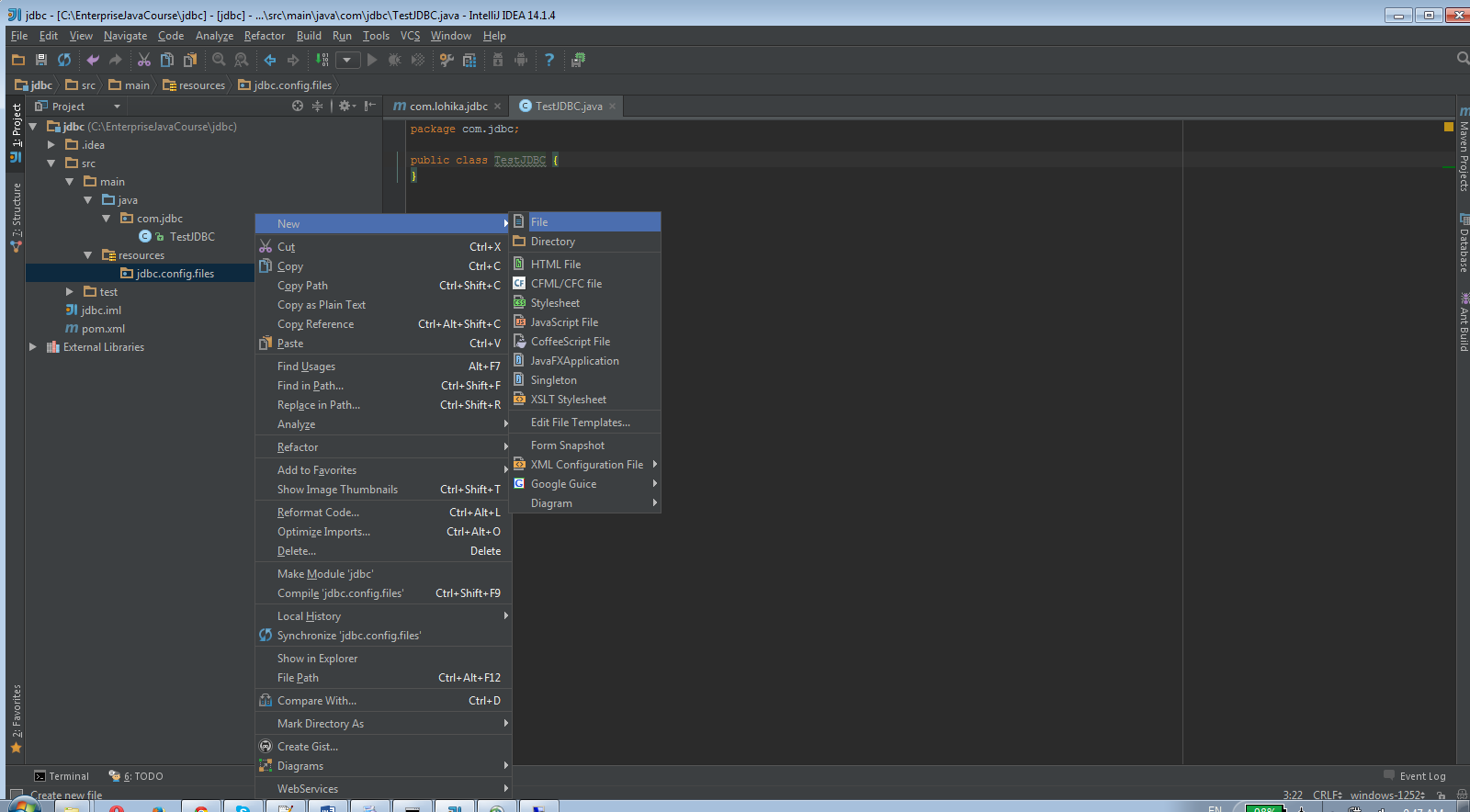
Кликаем правой кнопкой по папочке **resources**

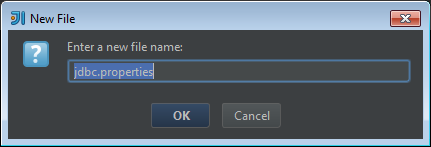
создаем пакет..например **jdbc.config.files**





Затем создаем сам **property file:**



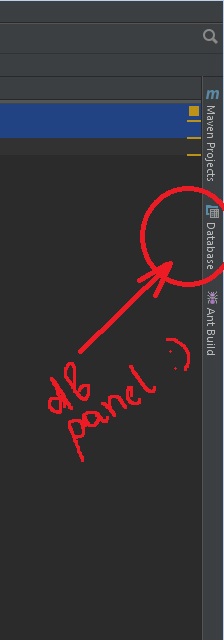


Содержимое **jdbc.properties**

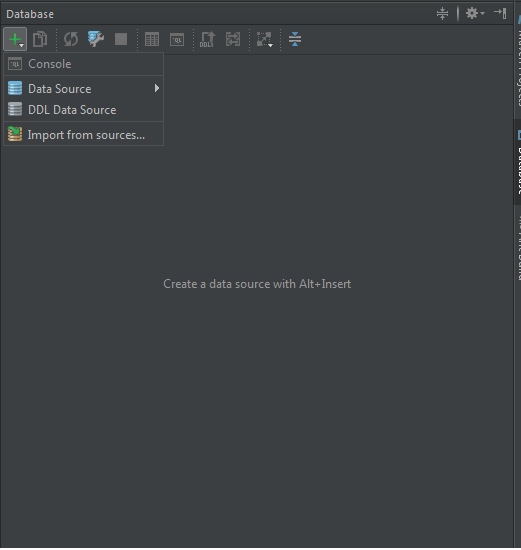
user.name=root  
user.password=123  
url=jdbc:mysql://localhost:3307/test\_jdbc\_db

Теперь попрошу немного отойти в сторонку, и посмотреть, откуда мы можем взять сам **url (jdbc:mysql://localhost:3307/test\_jdbc\_db)**

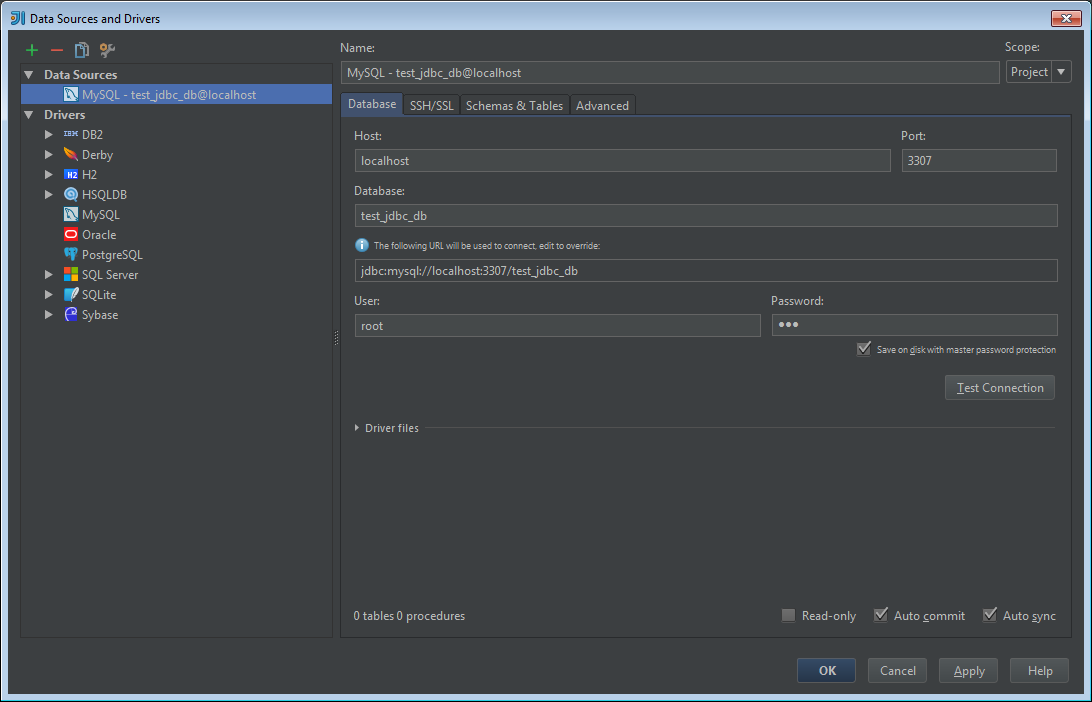
Посмотрите внимательно на панельки в **IDE – справа:**



Кликните на нее – найдите «+»



Затем выберите **DataSource -> MySQL**



Заполните все необходимые поля, исходя из того, что мы делали в пунктах [1 - 2]; в моем случае, это:

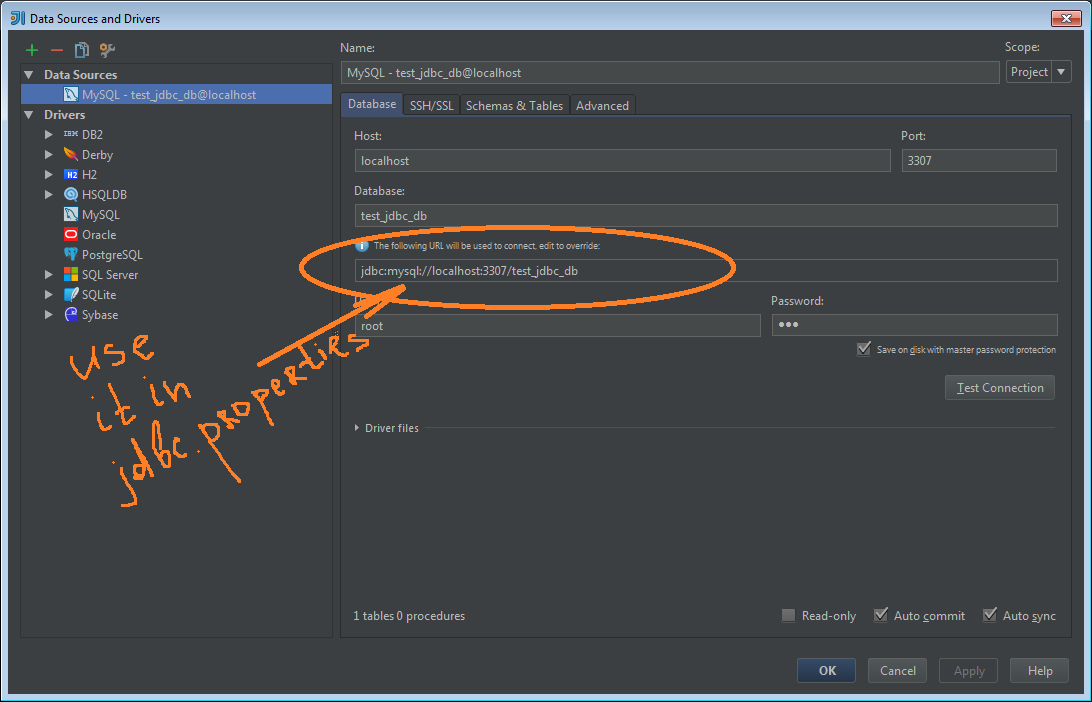
**Database** => **test\_jdbc\_db**

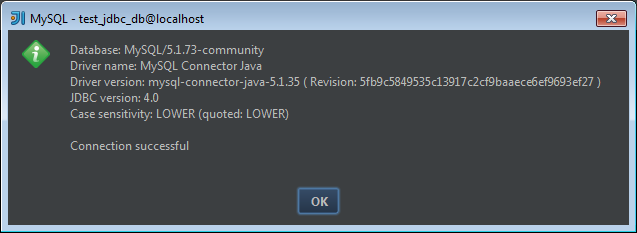
**User => root**

**Password => \*\*\* (**тот, который Вы указали при установке **MySQL**), для меня он определен, как 123)))

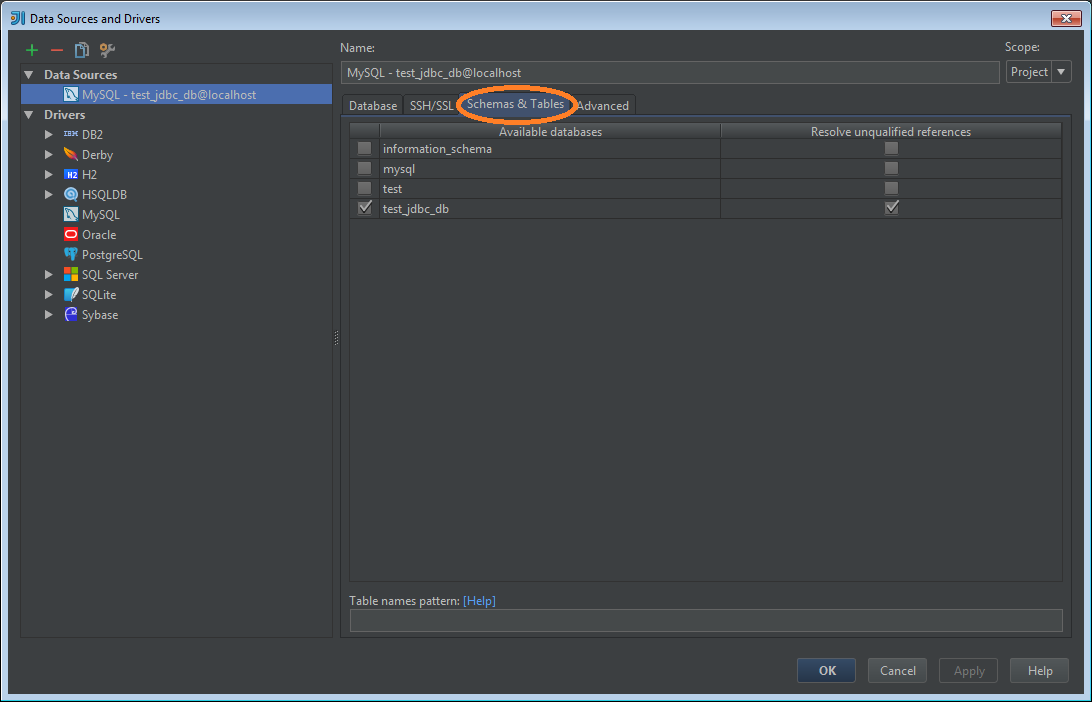
**Проверим коннект к базе – кликнем на «Test Connection» кнопку**

**Прошу обратить внимание, что url к базе, который мы будем использовать для коннекта уже создан/собран автоматически**



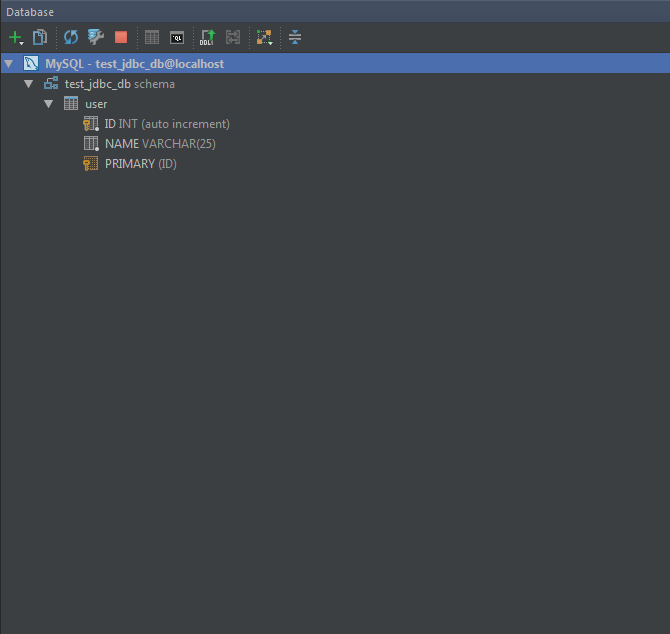


Рассмотрим доступные таблички, на этой же панельке найдем табу «Schemas && Tables»

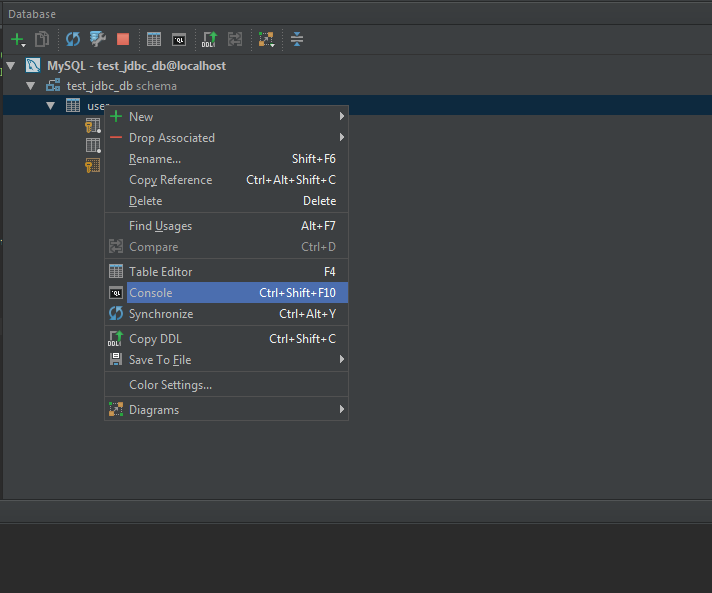


Заметьте, что галочка стоит напротив нашей схемки, кликнем на кнопочки **Apply -> ok**

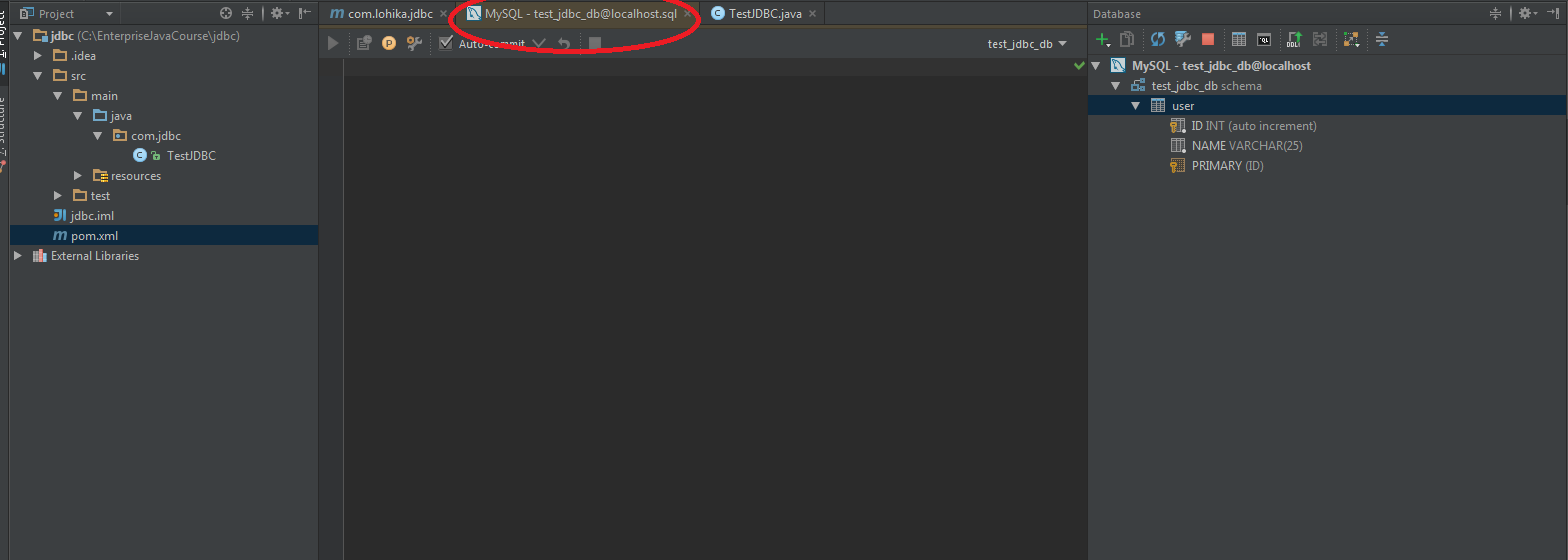
**После этого Вы имеете возможность делать запросы к БД напрямую, не используя MySQL Client console(как в пункте 2):**



Наведите на название Вашей таблички, затем щелкните правой кнопкой и выберите **«Console»**:



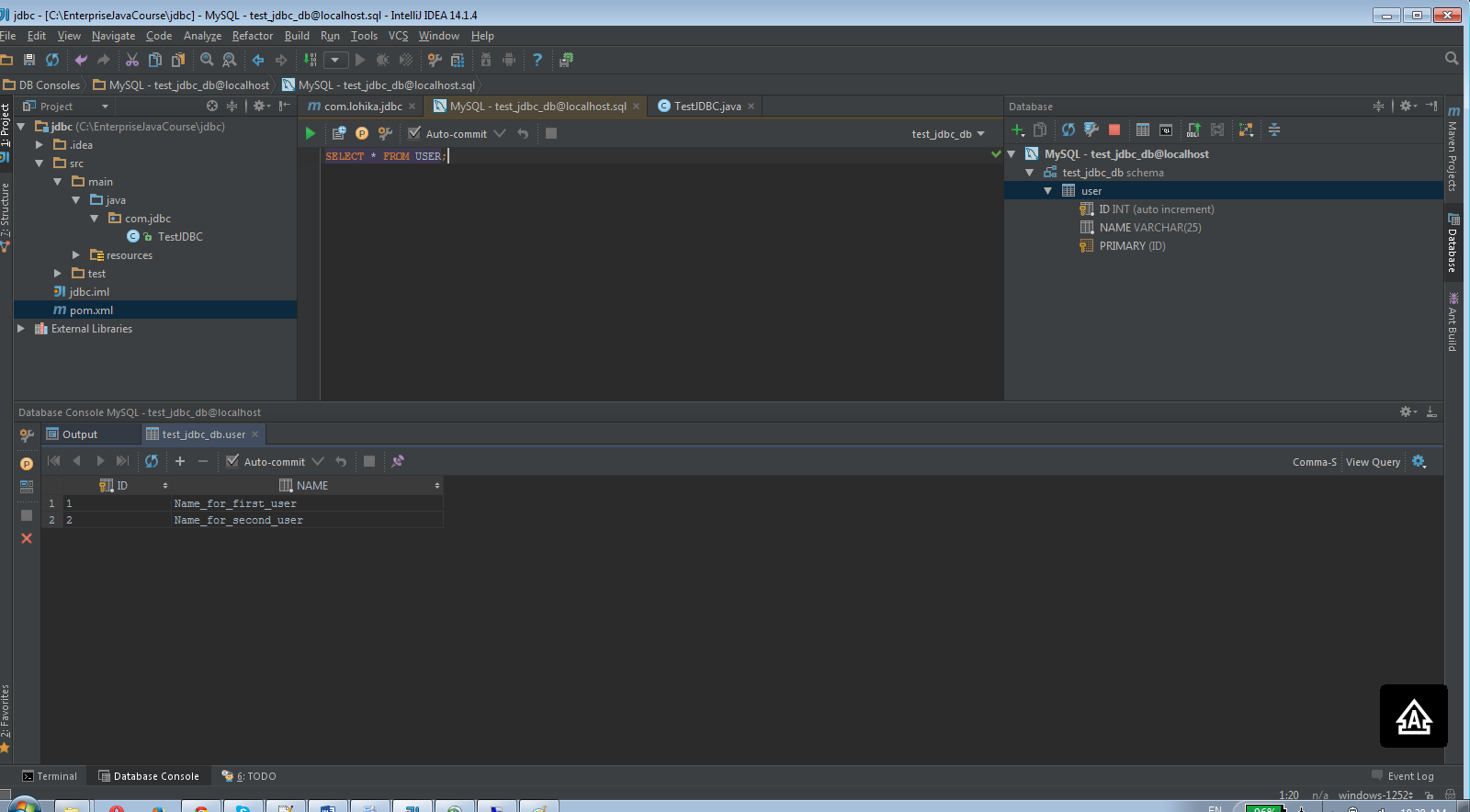
**После этого откроется консоль, где у Вас появится возможность выполнять запросы:**



В пределах этого окна, Вы можете выполнять любого типа, вида запросы – к Вашей схемке

**Например:**

**SELECT \* FROM USER;**



1. По сути у нас с Вами есть:

**- база данных**

**- jdbc.properties** file со следующими параметрами

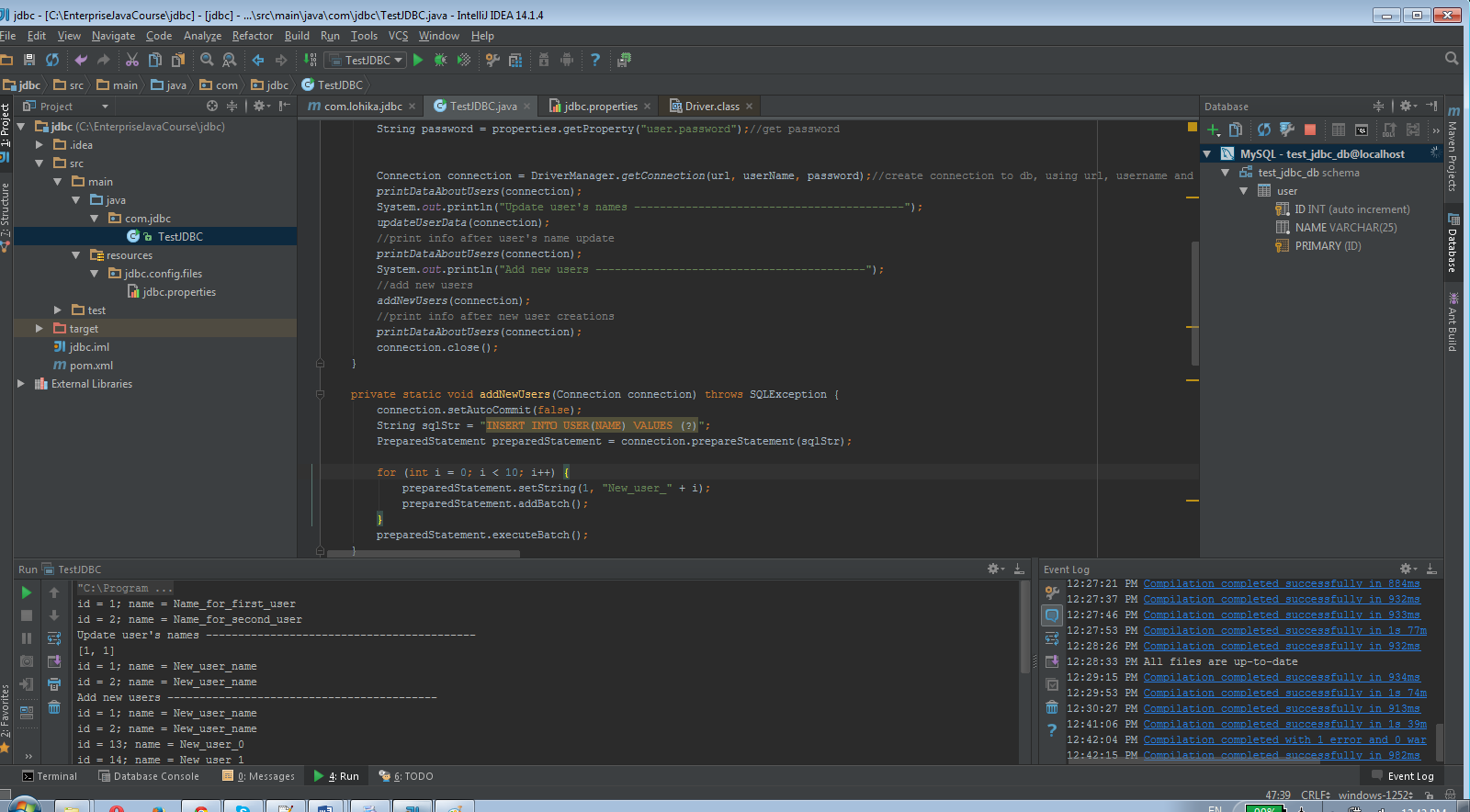
user.name=root  
user.password=123  
url=jdbc:mysql://localhost:3307/test\_jdbc\_db

**-** нужные **dependency**, подключенные посредством **pom.xml**

**- TestJDBC** класс **=** в пакете **jdbc**

**Поэтому, перейдем к написанию функционала:**

реализуем **пару методов,** позволяющих делать выборку, обновление, вставку новых данных в нашу табличку **user,** содержимое **– TestJDBC** класса

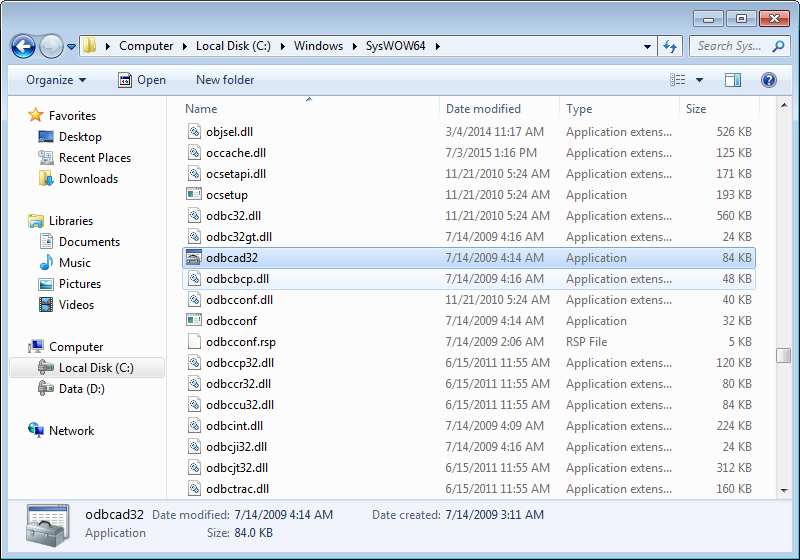


package com.jdbc;  
  
import java.io.IOException;  
import java.io.InputStream;  
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.PreparedStatement;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.sql.Statement;  
import java.util.Arrays;  
import java.util.Properties;  
  
public class TestJDBC {  
 private static String *USER\_INFO* = "id = %s; name = %s \n";  
  
 public static void main(String[] args) throws ClassNotFoundException, IOException, SQLException {  
 TestJDBC testJDBC = new TestJDBC();  
 Class.*forName*("com.mysql.jdbc.Driver");//register jdbc.properties driver  
 Properties properties = new Properties();  
 testJDBC.loadProps(properties);  
  
 String url = properties.getProperty("url");//get url from our jdbc.properties file  
 String userName = properties.getProperty("user.name");//get username  
 String password = properties.getProperty("user.password");//get password  
  
  
 Connection connection = DriverManager.*getConnection*(url, userName, password);//create connection to db, using url, username and pswrd from jdbc.properties file  
 *printDataAboutUsers*(connection);  
 System.*out*.println("Update user's names ------------------------------------------");  
 *updateUserData*(connection);  
 //print info after user's name update  
 *printDataAboutUsers*(connection);  
 System.*out*.println("Add new users ------------------------------------------");  
 //add new users  
 addNewUsers(connection);  
 //print info after new user creations  
 *printDataAboutUsers*(connection);  
 connection.close();  
 }  
  
 private static void addNewUsers(Connection connection) throws SQLException {  
 connection.setAutoCommit(false);  
 String sqlStr = "INSERT INTO USER(**NAME**) VALUES (?)";  
 PreparedStatement preparedStatement = connection.prepareStatement(sqlStr);  
  
 for (int i = 0; i < 10; i++) {  
 preparedStatement.setString(1, "New\_user\_" + i);  
 preparedStatement.addBatch();  
 }  
 preparedStatement.executeBatch();  
 }  
  
 private static void updateUserData(Connection connection) throws SQLException {  
 connection.setAutoCommit(false);  
 String updateSQLStr = "UPDATE USER SET **NAME**=? WHERE **ID**=?";  
 PreparedStatement preparedStatement = connection.prepareStatement(updateSQLStr);  
 preparedStatement.setString(1, "New\_user\_name");  
 preparedStatement.setString(2, "1");  
 preparedStatement.addBatch();  
  
 preparedStatement.setString(1, "New\_user\_name");  
 preparedStatement.setString(2, "2");  
 preparedStatement.addBatch();  
  
 int[] resultOfExecution = preparedStatement.executeBatch();  
 /\*  
 A number greater than or equal to zero -- indicates that the command was processed successfully and is an update count giving the number of rows in the database that were affected by the command's execution  
 A value of SUCCESS\_NO\_INFO -- indicates that the command was processed successfully but that the number of rows affected is unknown  
 A value of EXECUTE\_FAILED -- indicates that the command failed to execute successfully and occurs only if a driver continues to process commands after a command fails  
 \*/  
 System.*out*.println(Arrays.*toString*(resultOfExecution));  
  
 }  
  
 public void loadProps(Properties properties) throws IOException {  
 InputStream in = this.getClass().getClassLoader().getResourceAsStream("jdbc/config/files/jdbc.properties");  
 properties.load(in);  
 }  
  
 public static void printDataAboutUsers(Connection connection) throws SQLException {  
 Statement statement = connection.createStatement();  
 statement.execute("SELECT \* FROM USER");  
 ResultSet resultSet = statement.getResultSet();  
 while (resultSet.next()) {  
 System.*out*.printf(*USER\_INFO*, resultSet.getString(1), resultSet.getString(2));  
 }  
 }  
}

**Первая часть туториала – завершена, перейдем к описанию использования odbc драйвера**

1. В случае, если у Вас

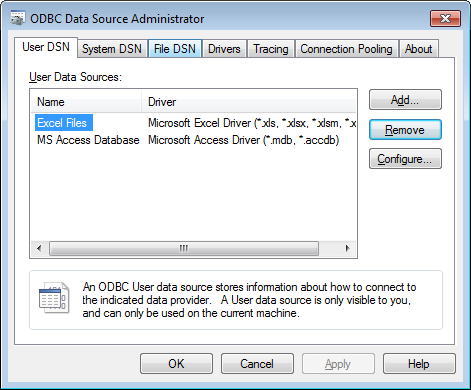
* 64 битная система – открываем **C:\Windows\SysWOW64**
* Если 32-битная система **C:\Windows\System32**



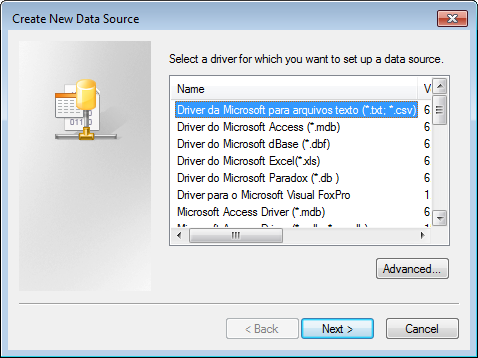
1. Create **excel doc**:

|  |  |  |
| --- | --- | --- |
| **id** | **name** | **salary** |
| 1 | x | 12 |
| 2 | y | 12 |
| 3 | z | 12 |

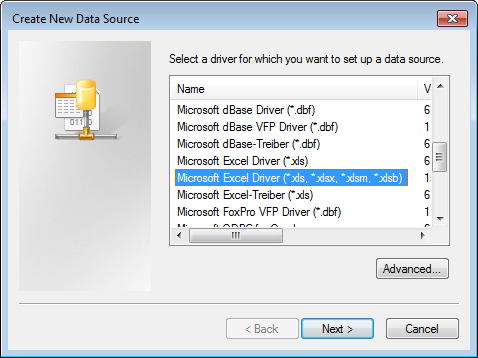
1. Сохраните его, допустим, как **UserExcel.xls**
2. Открываем **odbc manager wizard** – смотрите п.1



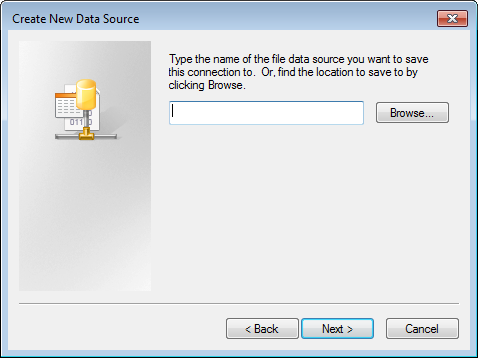
1. Открываем табку **FileDSN -> Add →**



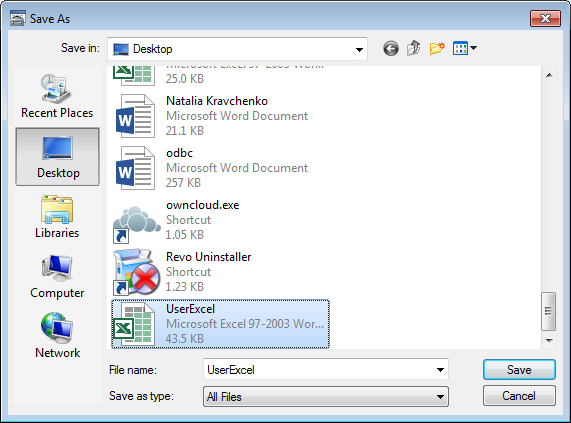
**Выбираем**

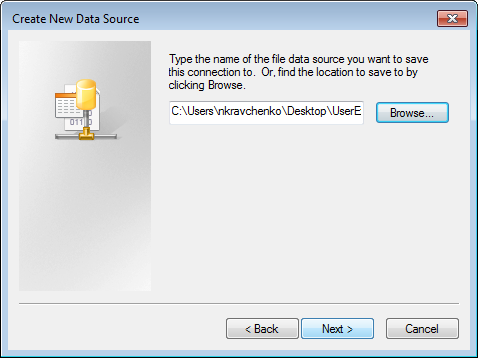


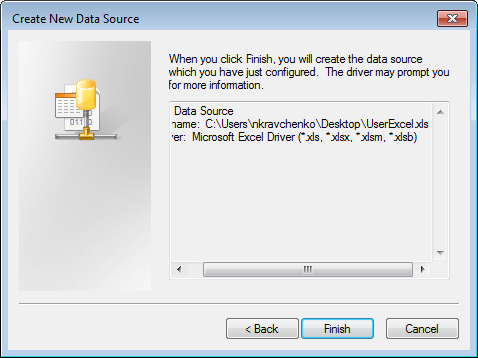
Затем нажимаем **Next**



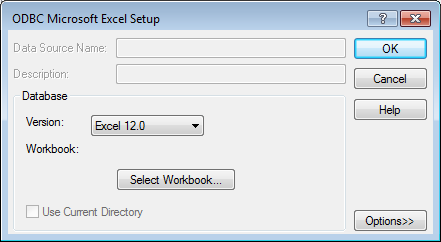
**Выбираем нашу «базу»(табличный source) - excel документ**



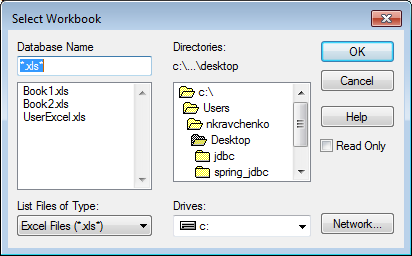




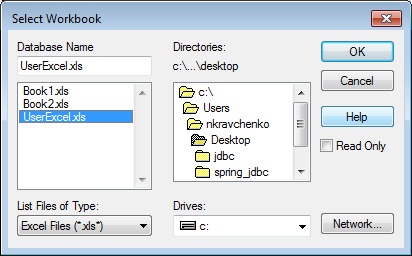
Нажимаем **Finish**



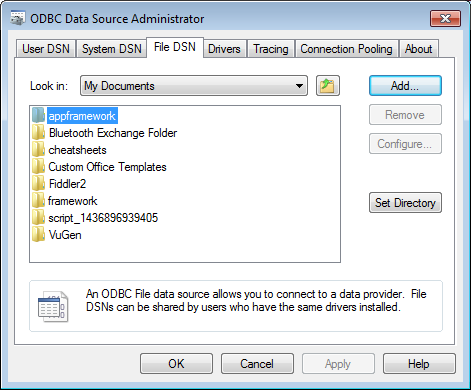
Затем просто кнопочку **«ок»**



Выбираем из списка **xls –** интересующий нас **UserExcel.xls**

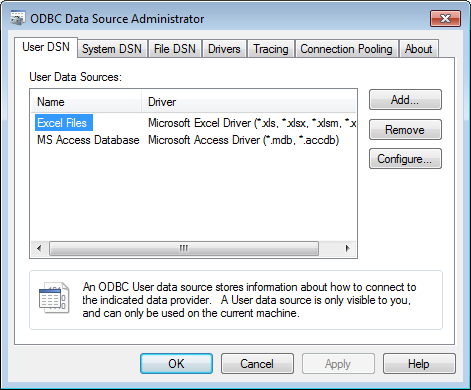


Затем снова **«ок»**



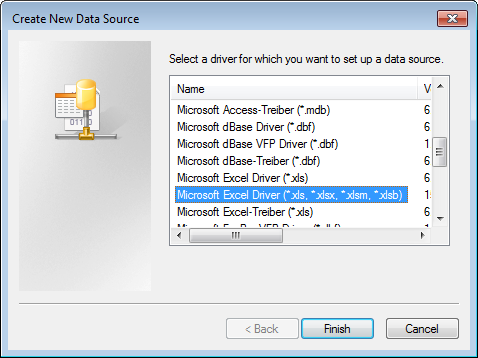
После добавления **FileDSN** создадим **UserDSN**

**Переходим на вкладку User DSN**

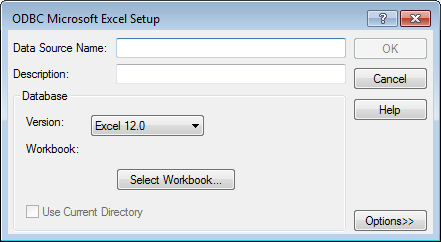


Нажимаем кнопочку **«Add»**

Выбираем, интересующий нас **xls driver**



Затем кнопочку **«Finish»**

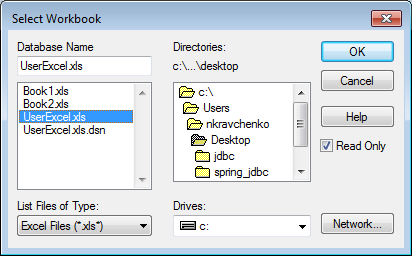


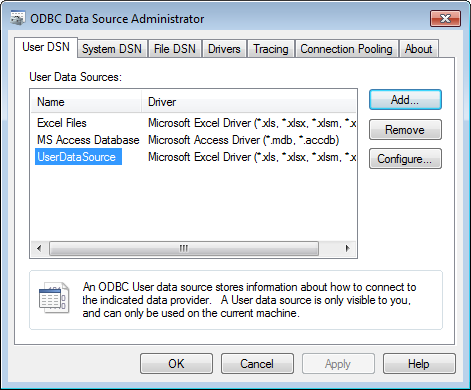
Указываем в качестве **– Data Source Name,** интуитивно-понятное название

**Например: UserDataSource**

**Description: любой**

Затем кликаем **“Select Workbook” ->** выбираем наш добавленный ранее workbook





Затем **«ок»**

1. **Переходим в IDE создаем класс по-аналогии с** TestJDBC

package com;  
  
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.PreparedStatement;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.sql.Statement;  
  
public class TestOdbc {  
 public static void main(String[] args) throws ClassNotFoundException, SQLException {  
 Class.*forName*("sun.jdbc.odbc.JdbcOdbc");  
 Connection connection = DriverManager.*getConnection*("jdbc:odbc:UserDataSource");  
 Statement statement = connection.createStatement();  
 ResultSet resultSet = statement.executeQuery("select [Sheet1$].id from [Sheet1$]");  
 while (resultSet.next()) {  
 System.*out*.println(resultSet.getString(1));  
 }  
  
 PreparedStatement preparedStatement = connection.prepareStatement("insert into [Sheet1$] values(?,?,?)");  
 preparedStatement.setObject(1,"12");  
 preparedStatement.setObject(2,"13");  
 preparedStatement.setObject(3,"67");  
 preparedStatement.executeUpdate();  
 connection.close();  
  
 }  
}

**В случае, если возникает ошибка**

**Exception in thread "main" java.sql.SQLException: [Microsoft][ODBC Excel Driver] Operation must use an updateable query.**

**at sun.jdbc.odbc.JdbcOdbc.createSQLException(JdbcOdbc.java:6964)**

**at sun.jdbc.odbc.JdbcOdbc.standardError(JdbcOdbc.java:7121)**

**at sun.jdbc.odbc.JdbcOdbc.SQLExecute(JdbcOdbc.java:3156)**

**at sun.jdbc.odbc.JdbcOdbcPreparedStatement.execute(JdbcOdbcPreparedStatement.java:215)**

**at sun.jdbc.odbc.JdbcOdbcPreparedStatement.executeUpdate(JdbcOdbcPreparedStatement.java:137)**

**at com.TestOdbc.main(TestOdbc.java:24)**

**at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)**

**at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)**

**at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)**

**at java.lang.reflect.Method.invoke(Method.java:606)**

**at com.intellij.rt.execution.application.AppMain.main(AppMain.java:140)**

**Проверьте, что снята галочка – с “Read only” в настройках User DSN**

