

БАЗИ ДАНИХ У JAVA-ДОДАТКАХ

У 1996 р. компанія Sun Microsystems випустила першу версію прикладного інтерфейсу API для організації доступу з програм Java до баз даних JDBC. Цей інтерфейс дозволяє з'єднуватися з базою даних, запитувати і оновлювати дані за допомогою мови структурованих запитів (Structured Query Language — SQL).

1 Необхідне програмне забезпечення

Встановіть СУБД MySQL Community Server (<http://dev.mysql.com/downloads/mysql/>).

Встановіть MySQL Workbench, якщо він не увійшов до базової комплектації MySQL.

Встановіть JDBC драйвер для MySQL, якщо його не було встановлено при інсталяції СУБД (<http://dev.mysql.com/downloads/connector/j/>). У випадкові використання Linux скопіюйте відповідний jar-файл (на момент написання цього тексту — mysql-connector-java-5.1.36-bin.jar) до папки /usr/share/java.

2 Створення проекту з підтримкою JDBC в IntelliJ IDEA

2.1 Створіть проект за допомогою IntelliJ IDEA.

2.2 Виберіть пункт меню File → Project Structure. У відповідному вікні виберіть пункт Libraries (рис. 1).

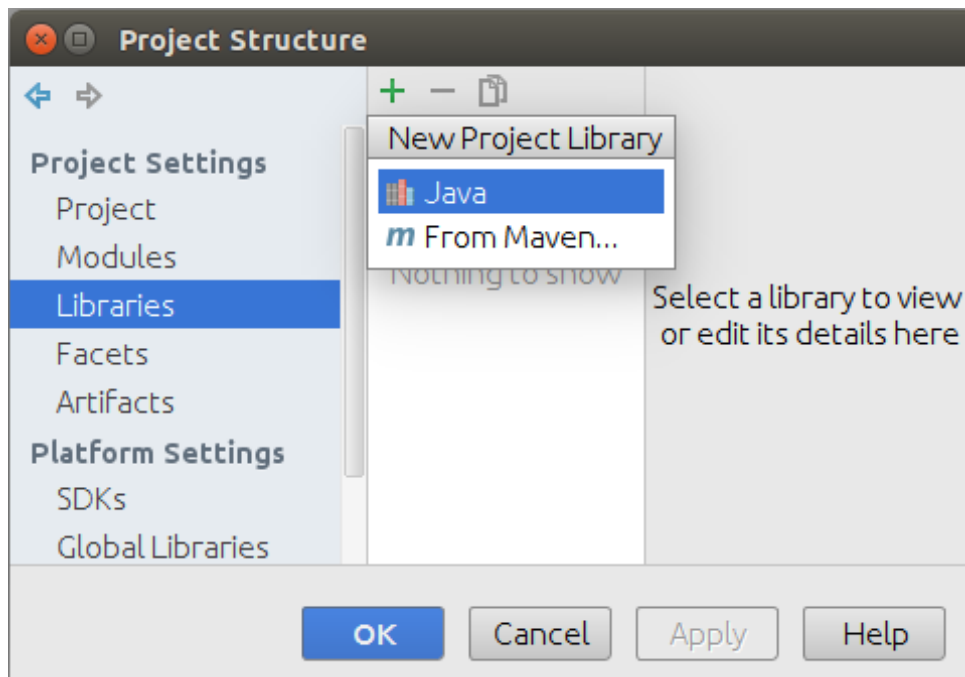


Рисунок 1 — Під'єднання бібліотек до проекту

Натисніть піктограму “+”. Зі списку New Project Library виберіть пункт Java. У вікні Select Library Files виберіть JDBC — драйвер. Для Linux шлях до драйвера повинен бути /usr/share/java (рис. 2).

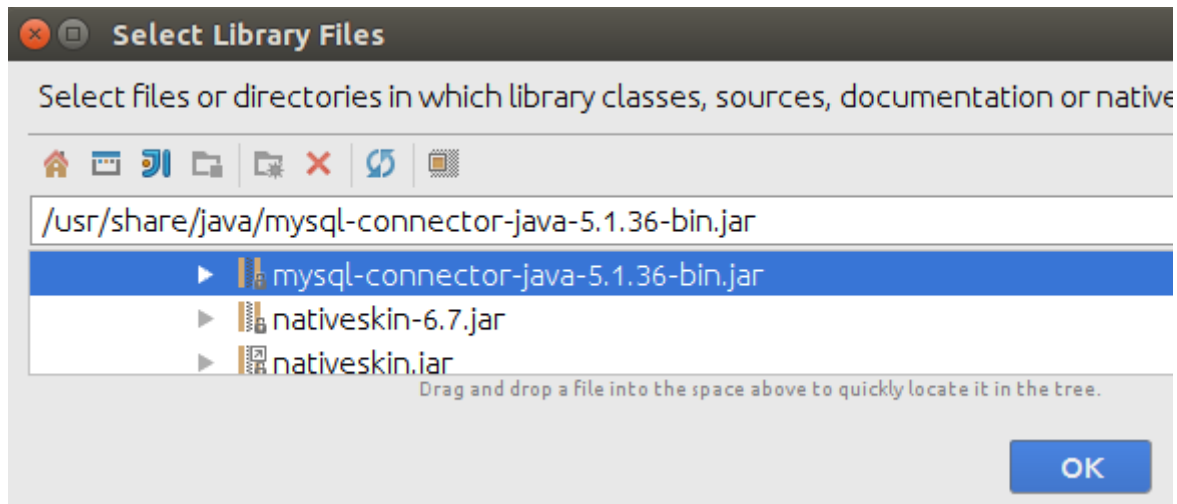


Рисунок 2 — Під'єднання драйвера JDBC для MySQL

<http://dev.mysql.com/doc/workbench/en/wb-table-editor-columns-tab.html>

3 Створення таблиці бази даних

Операції із таблицями бази даних можна здійснювати безпосередньо за допомогою консолі MySQL. На даному етапі скористаємося для наочності MySQL Workbench.

Завантажте MySQL Workbench. При першому запускові натисніть піктограму “+” (рис. 3). Ця дія необхідна для створення та налаштування під'єднання до бази даних.

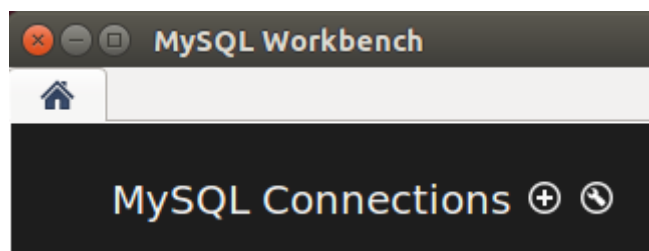


Рисунок 3 — Елемент стартового екрану SQL Workbench

Заповніть відповідні поля вікна Setup New Connection:

- Connection Name: задайте довільне ім'я, наприклад, MyConnection;
- Connection Method - Standard (TCP/IP);
- Hostname: 127.0.0.1, Port: 3306 (це стандартні значення);
- Username: ім'я користувача. Залежить від налаштувань MySQL;
- Password: пароль; можна не задавати його тут, тоді слід буде вводити пароль на вимогу;
- Default Schema: ім'я бази даних. Можна залишити поле порожнім і вибрати базу даних пізніше.

Натиснувши кнопку Test Connection можна перевірити правильність налаштування з'єднання (рис. 4).

На цьому налаштування з'єднання із сервером MySQL завершено і можна перейти до створення та наповнення таблиці із даними.

Натисніть кнопку OK.

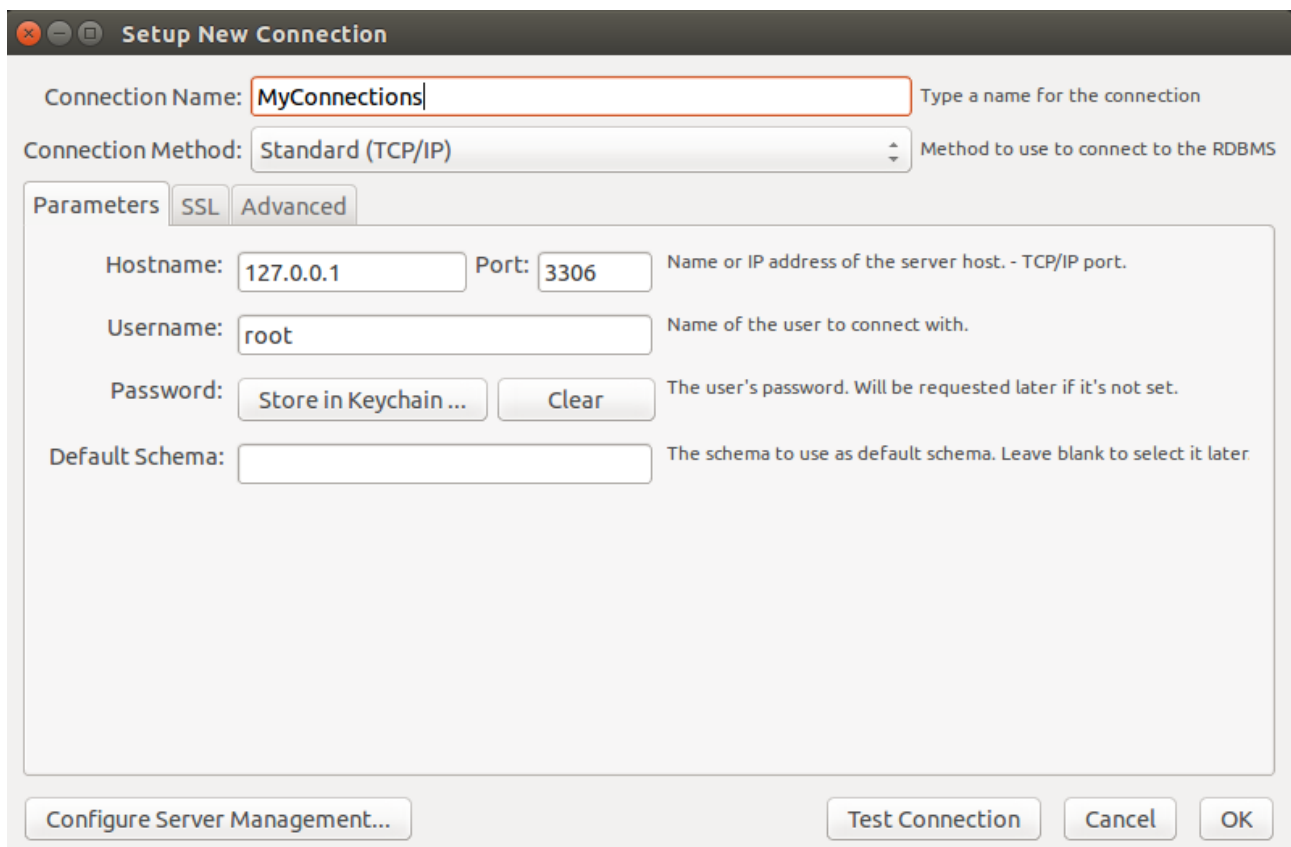


Рисунок 4 — Налаштування з'єднання із сервером MySQL

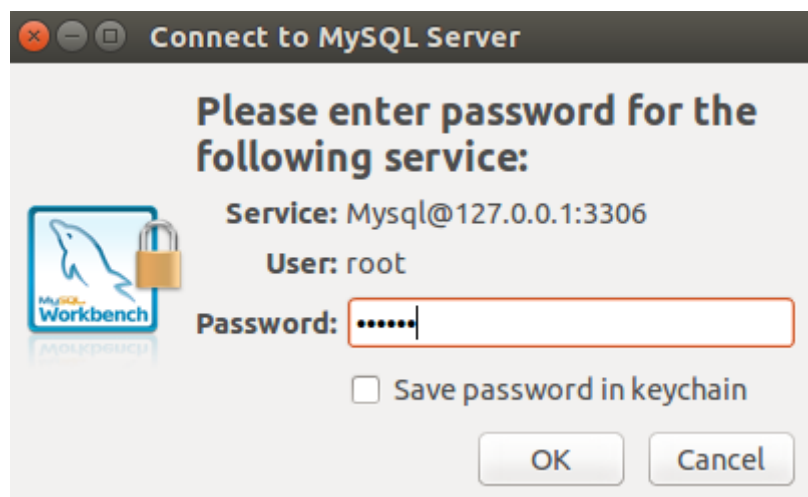


Рисунок 5 — Аутентифікація при перевірці з'єднання

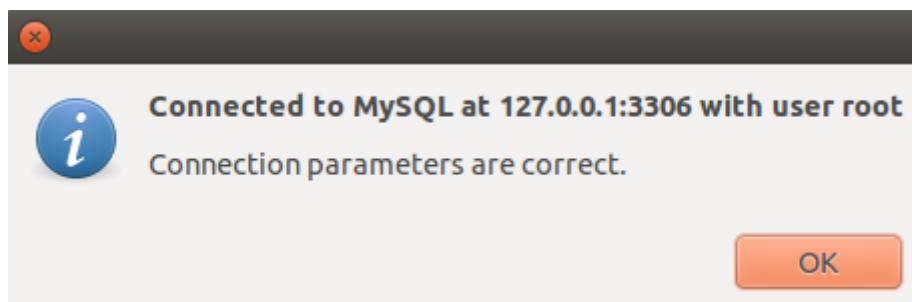


Рисунок 6 — Підтвердження правильності налаштувань з'єднання

Після створення з'єднання стартовий екран MySQL Workbench міститиме піктограму у вигляді прямокутника із відповідним іменем (рис. 7).

Клікніть по піктограмі. Після проходження користувачем аутентифікації відкриється вікно MySQL Workbench (рис. 8).

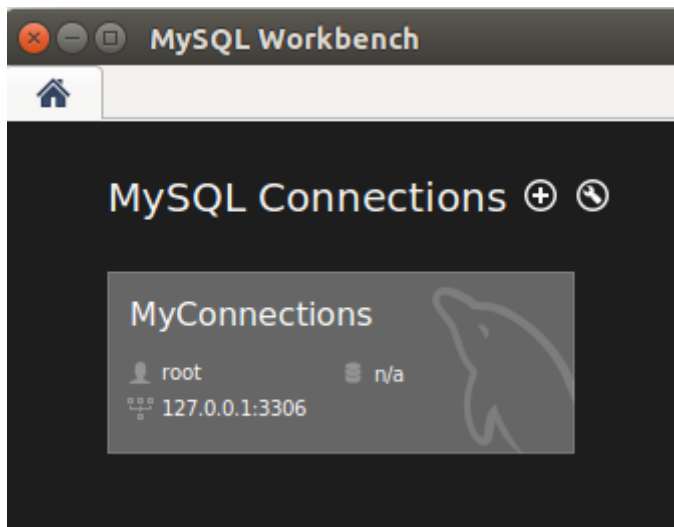


Рисунок 7 — Нове з'єднання

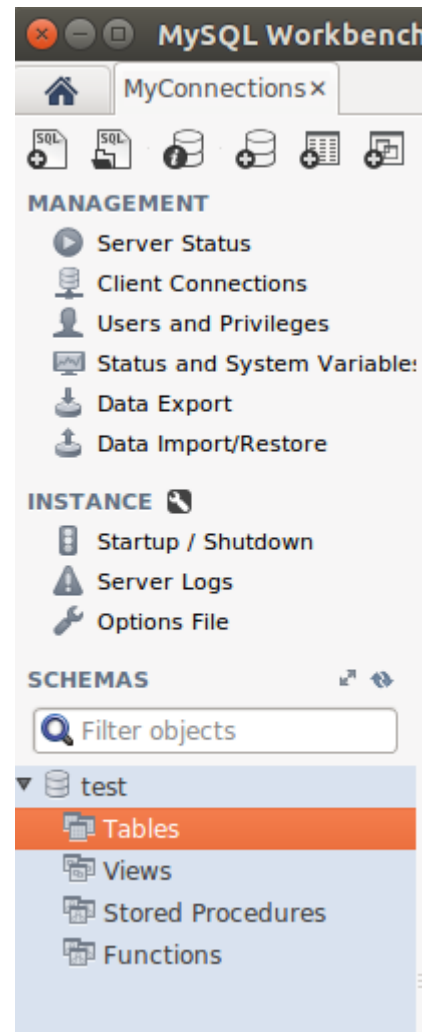


Рисунок 8 — Таблиці MySQL

Розкрийте тестову базу даних test, натиснувши на трикутник зліва від імені бази даних. Виберіть пункт Tables, натисніть праву кнопку миші і, вибравши Create Table, перейдіть до створення таблиці (рис. 9). Задайте ім'я таблиці та оголошіть її поля.

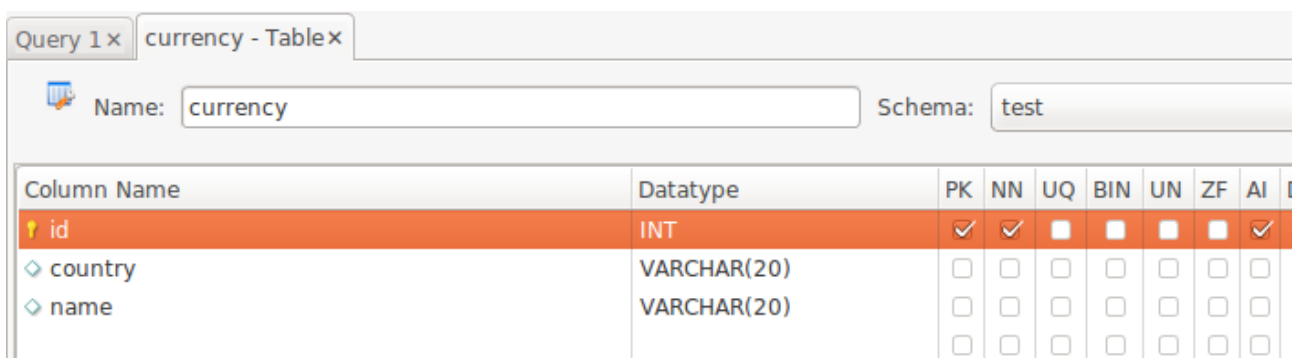


Рисунок 9 — Оголошення полів таблиці

Призначення флагів:

PK — Primary Key;
NN — Not Null;
UQ — Unique Key;
BIN — Binary
UN — Unsigned;
ZF — Zero-Filled;
AI — Auto Increment.

Для додавання чергового поля клікніть по рядкові під поточним. Після завершення редагування натисніть кнопку Apply. При цьому відкриється вікно підтвердження застосування SQL-скрипта до бази даних (рис. 10). Натисніть кнопку Apply. Успішність виконання скрипта підтверджує відповідне вікно (рис. 11).

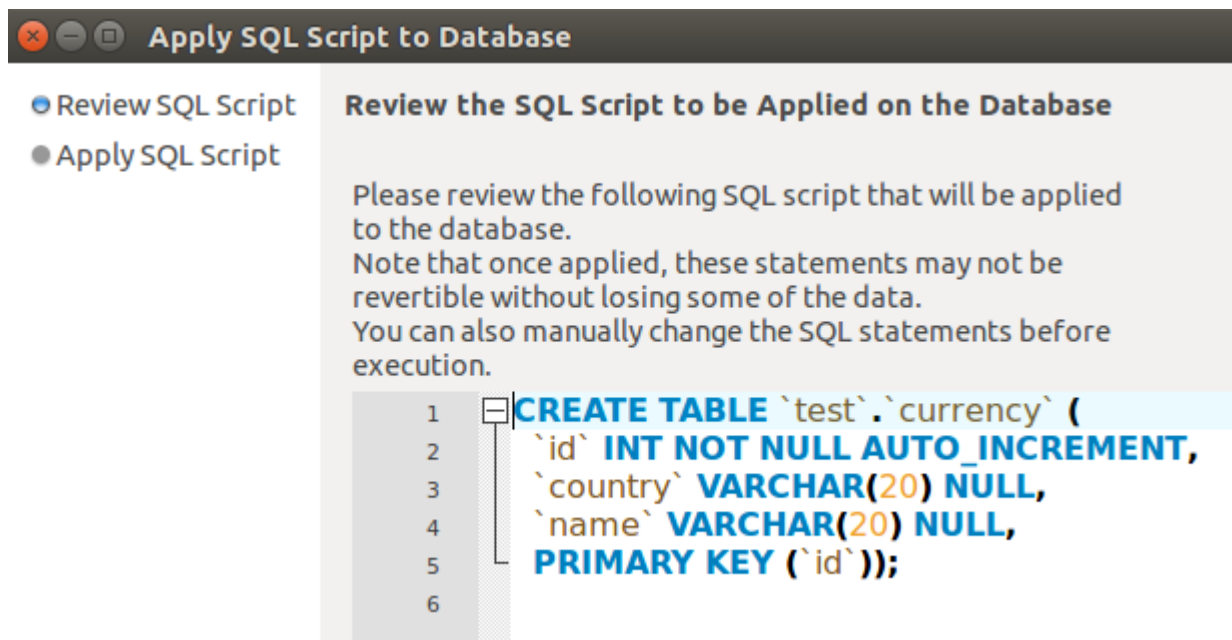


Рисунок 10 — Підтвердження застосування SQL-скрипта

```
CREATE TABLE `test`.`currency` (`id` INT NOT NULL AUTO_INCREMENT,  
`country` VARCHAR(20) NULL, `name` VARCHAR(20) NULL, PRIMARY KEY (`id`));
```

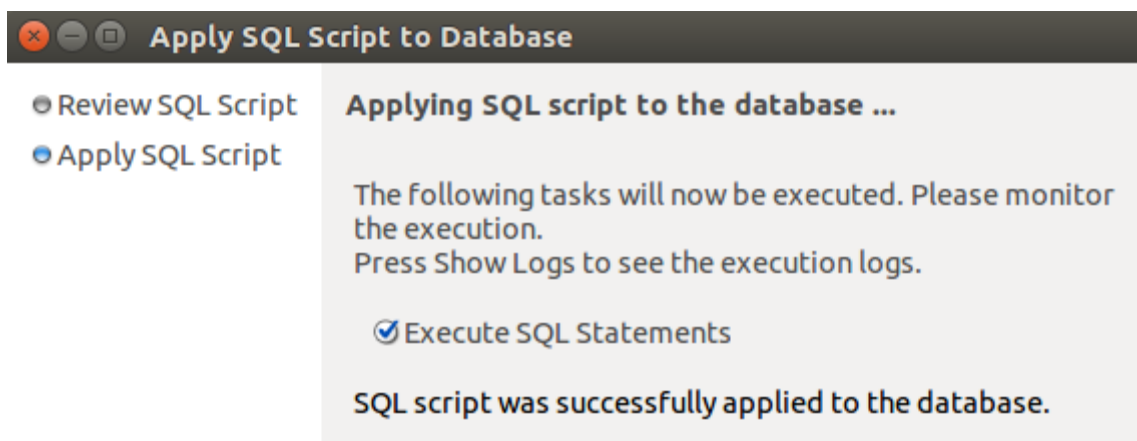


Рисунок 11 — Підтвердження успішного застосування SQL-скрипта

4 Наповнення таблиці бази даних

Виконайте наступний запит до таблиці `test`.`currency` (рис. 12):

```
SELECT * FROM `test`.`currency`
```

Заповніть поля таблиці у відповідності до рис. 13. У кінці натисніть піктограму, що позначає блискавку.

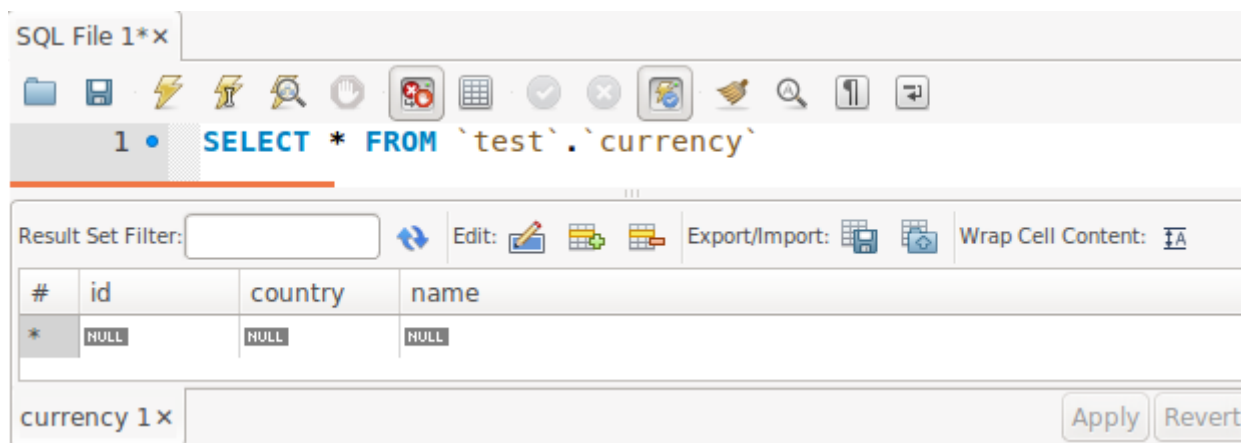


Рисунок 12 — Виконання запиту

Використовуйте подвійний клік лівої кнопки миші при наведеному курсорові на потрібне поле таблиці. Завершує редагування комірки кнопка клавіатури Enter.

Перехід між полями можна здійснювати також за допомогою кнопки Tab.

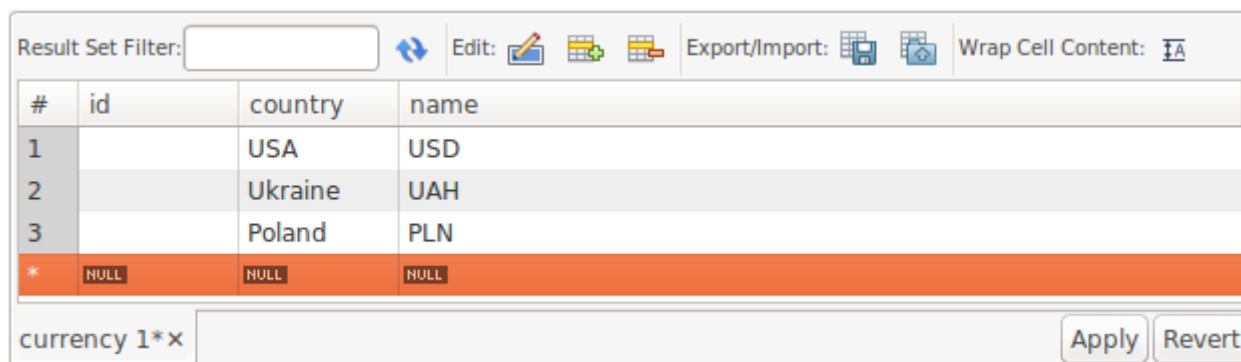


Рисунок 13 — Наповнення полів таблиці даними

Після завершення натисніть кнопку Apply. При цьому будуть виконані SQL-інструкції, про що свідчить підтверджуюче вікно (рис. 14):

```
INSERT INTO `test`.`currency` (`country`, `name`) VALUES ('USA', 'USD');  
INSERT INTO `test`.`currency` (`country`, `name`) VALUES ('Ukraine', 'UAH');  
INSERT INTO `test`.`currency` (`country`, `name`) VALUES ('Poland', 'PLN');
```

Прийміть зміни.

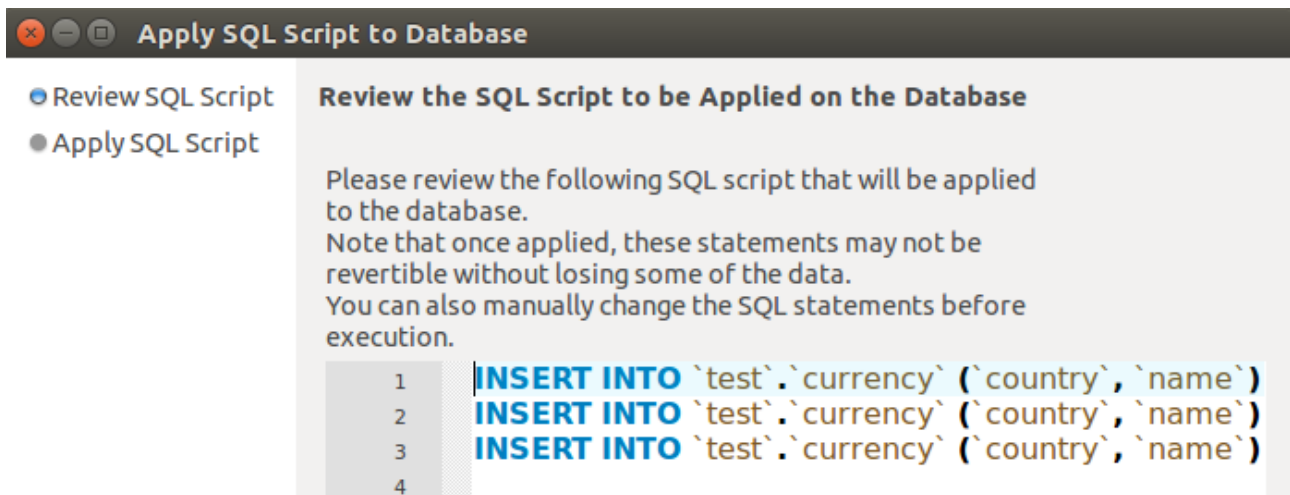


Рисунок 14 — Підтвердження застосування SQL-скрипта

5 Приклад

```
import java.sql.*;
class MysqlCon {
    public static void main(String args[]) {
        Connection connection = null;
        try {
            connection = DriverManager.
                getConnection("jdbc:mysql://localhost:3306/test", "root", "password");
        } catch (SQLException e) {
            System.out.println("Connection Failed!");
            e.printStackTrace();
            return;
        }
        if (connection != null) {
            System.out.println("Connection Established\n");
        } else {
            System.out.println("Failed to make a connection");
        }
        try {
            Statement stmt = connection.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM currency");
            while (rs.next())
                System.out.println(
                    rs.getInt(1) + " " +
                    rs.getString(2) + " " +
                    rs.getString(3));
            connection.close();
        } catch (SQLException e) {
            System.out.println(e);
        }
    }
}
```

Розглянемо приклад детальніше. Клас *DriverManager* — основний сервіс для

управління набором JDBC драйверів. Зверніть увагу на те, що додаткам більше не потрібно завантажувати JDBC драйвери у явному вигляді за допомогою `Class.forName()`:
(<http://docs.oracle.com/javase/7/docs/api/java/sql/DriverManager.html>).

Виклик методу `getConnection()` класу `DriverManager` ініціює спробу `DriverManager` знайти відповідний драйвер між драйверами, завантаженими при ініціалізації та завантаженими у явному вигляді за допомогою того самого завантажувача класів, що використовують аплет чи додаток:

*public static [Connection](#) getConnection(
[String](#) url, [Properties](#) info) throws [SQLException](#)*

Намагається встановити зв'язок у відповідності із заданим URL бази даних, вибравши відповідний драйвер з-поміж зареєстрованих JDBC драйверів.

Параметри:

url – URL бази даних у формі *jdbc:subprotocol:subname* ;

info – список довільних пар типу *string* дескриптор/значення у якості аргументів з'єднання; зазвичай повинні бути подані принаймні властивості "user" та "password".

Як результат повертається *Connection* до відповідного URL.

Винятки: [SQLException](#) – у випадкові виникнення помилки доступу до бази даних.

Interface Connection – з'єднання (сесія) зі вказаною базою даних. Виконання інструкцій SQL та отримання результату здійснюють за допомогою *Connection*. Детальніше тут: <http://docs.oracle.com/javase/7/docs/api/java/sql/Connection.html> .

[Statement](#) createStatement() throws [SQLException](#)

Створює об'єкт *Statement* для надсилання SQL-виразів до бази даних. SQL вирази без параметрів зазвичай виконують за допомогою об'єктів *Statement*. Якщо SQL-вираз виконують багато разів, доцільно скористатися об'єктом *PreparedStatement*.

Набір результатів *ResultSet*, створений за допомогою об'єкта *Statement*, має тип `TYPE_FORWARD_ONLY` з рівнем паралелізму `CONCUR_READ_ONLY`.

Повертає: новий об'єкт *Statement*.

Винятки: [SQLException](#) - у випадкові виникнення помилки доступу до бази даних або якщо метод застосовано над закритим з'єднанням.

[ResultSet](#) executeQuery([String](#) sql) throws [SQLException](#)

<http://docs.oracle.com/javase/7/docs/api/java/sql/Statement.html#executeQuery%28java.lang.String%29>

Виконує даний SQL вираз та повертає об'єкт *ResultSet*. Метод не може бути використано над *PreparedStatement* та *CallableStatement*.

Параметри: *sql* - SQL-вираз.

Повертає: об'єкт *ResultSet*, що містить дані, які відповідають поточному запиту; не може бути *null*.

Винятки:

[SQLException](#) - у випадкові виникнення помилки доступу до бази даних, метод застосовано над закритим *Statement*, результатом даного SQL вираз є щось інше, ніж об'єкт *ResultSet*, метод викликано над *PreparedStatement* або *CallableStatement*.

[SQLTimeoutException](#) – якщо драйвер визначив, що таймаут, встановлений за допомогою методу *setQueryTimeout*, перевищено, і спробував скасувати поточний *Statement* .

6 Основні методи класу ResultSet

<http://docs.oracle.com/javase/7/docs/api/java/sql/ResultSet.html>

```
public boolean absolute(int row) throws SQLException;
```

переміщує курсор на задане число рядків від початку, якщо параметр row — додатній, і від кінця — якщо від'ємний.

```
public void afterLast() throws SQLException;
```

переміщує курсор у кінець результуючого набору, за останній рядок.

```
public void beforeFirst() throws SQLException;
```

переміщує курсор на початок результуючого набору, перед першим рядком.

```
public void deleteRow() throws SQLException;
```

видаляє поточний рядок із результуючого набору і бази даних.

```
public ResultSetMetaData getMetaData() throws SQLException;
```

надає об'єкт метаданих для даного ResultSet. Клас ResultSetMetaData містить інформацію про результуючі таблиці, таку як кількість стовпців, їх заголовки тощо.

```
public int getRow() throws SQLException;
```

повертає номер поточного рядка.

```
public Statement getStatement() throws SQLException;
```

повертає екземпляр Statement, котрий створив даний результуючий набір.

```
public boolean next() throws SQLException;  
public boolean previous() throws SQLException;
```

дозволяють переміститися у результуючому наборі на один рядок вперед або назад. У результуючому наборі курсор встановлюється перед першим рядком, тому перше звернення до методу next() викликає позиціонування на перший рядок. Методи повертають true, якщо залишається рядок для подальшого переміщення. Якщо рядків для обробки більше немає, повертається false. Якщо відкритий потік InputStream для попереднього рядка, він закривається. Також очищається ланцюжок попереджень SQLWarning.

```
public void close() throws SQLException
```

здійснює негайне закриття ResultSet вручну. Зазвичай цього не потрібно, так як закриття Statement, пов'язаного з ResultSet, автоматично закриває ResultSet. Не всі розробники JDBC-драйверів дотримуються цих конвенцій, наприклад, драйвер Oracle самостійно не закриває ResultSet'и.

7 Основні методи класу Connection

<http://docs.oracle.com/javase/7/docs/api/java/sql/Connection.html>

```
public void close() throws SQLException;
```

дозволяє уручну звільнити ресурси такі, як мережеві з'єднання і блокування бази даних, пов'язані з даним об'єктом Connection (метод автоматично викликається при збірці сміття).

```
public Statement createStatement() throws SQLException;  
public Statement createStatement(int type, int concur) throws SQLException;
```

створює об'єкт Statement, пов'язаний з сеансом Connection. Версія без аргументів створює об'єкт Statement, для якого екземпляри ResultSet мають тип тільки для читання і переміщення в прямому напрямку.

```
public boolean getAutoCommit() throws SQLException;  
public void setAutoCommit(boolean ac) throws SQLException;
```

за замовчуванням об'єкти Connection знаходяться в режимі автозавершення. У цьому режимі кожна команда завершується одразу після виконання. Може виявитися кращим вручну завершити серію команд як єдину транзакцію. У цьому випадку метод setAutoCommit() використовують для відключення автозавершення. Потім, після виконання команд, викликають commit() або rollback(), залежно від успіху чи неуспіху транзакції. У режимі автозавершення команда завершується, коли вона виконана, або виконується наступна команда, в залежності від того, що відбудеться раніше. Команда, яка повертає ResultSet, виконана після екстракції останнього рядка або закриття об'єкта ResultSet. Якщо команда повертає множинні результуючі набори, завершення відбувається після екстракції останнього рядка останнього об'єкта ResultSet.

```
public void commit() throws SQLException;
```

метод робить постійними зміни, створені усіма командами, пов'язаними з даним з'єднанням і виконаними після останньої команди завершення або відкату транзакції. Використовувати його слід тільки при відключеному автозавершенні. Він не завершує зміни, зроблені командами, які пов'язані з іншими об'єктами Connection.

```
public String getCatalog() throws SQLException;  
public void setCatalog(String catalog) throws SQLException;
```

Якщо драйвер підтримує каталоги, то setCatalog() використовують для вибору підпростору бази даних із заданим ім'ям каталогу. Якщо драйвер каталоги не підтримує, запит ігнорується.

```
public DatabaseMetaData getMetaData() throws SQLException;
```

клас DatabaseMetaData надає методи, що описують таблиці бази даних, підтримку SQL, stored-процедури та інші відомості, що стосуються бази даних і даного Connection, які не стосуються безпосередньо виконання команд і отримання результуючих наборів даних. Метод створює екземпляр класу DatabaseMetaData для даного Connection.

```
public SQLWarning getWarnings() throws SQLException;
```

Повертає перше попередження зі списку, пов'язаного з даним об'єктом *Connection*.

7 Основні методи класу *Statement*

<http://docs.oracle.com/javase/7/docs/api/java/sql/Statement.html>

```
public void addBatch(String sql) throws SQLException;
```

додає задану команду SQL до поточного пакету команд.

```
public void cancel() throws SQLException;
```

у багатопоточному середовищі за допомогою цього методу можна зажадати припинення обробки, пов'язаної з даним *Statement*. У цьому сенсі метод аналогічний до методу *stop()* для об'єктів *Thread*.

```
public boolean execute(String sql) throws SQLException;  
public ResultSet executeQuery(String sql) throws SQLException;  
public int executeUpdate(String sql) throws SQLException;
```

виконує *Statement*, передаючи базі даних заданий SQL-рядок. Перший метод, *execute()* дозволяє виконати *Statement*, коли не відомо заздалегідь, є SQL-рядок запитом чи оновленням. Метод повертає *true*, якщо команда створила результуючий набір. Метод *executeQuery()* використовують для виконання запитів (на отримання даних). Він повертає для обробки результуючий набір. Метод *executeUpdate()* використовують для виконання оновлень. Він повертає кількість оновлених рядків.

```
public int[ ] executeBatch(String sql) throws SQLException;
```

надсилає базі даних пакет SQL-команд для виконання. Повертає масив чисел, що описують кількість рядків, опрацьованих кожною командою SQL.

```
public ResultSet getResultSet() throws SQLException;
```

повертає поточний *ResultSet*. Для кожного результату його слід викликати тільки один раз. Метод не потрібно викликати після звернення до *executeQuery()*, що повертає єдиний результат.

```
public void close() throws SQLException;
```

уручну закриває об'єкт *Statement*. Зазвичай цього не потрібно, оскільки *Statement* автоматично закривається при закритті пов'язаного з ним об'єкта *Connection*.

Для виконання маніпуляцій з даними (оновлення, додавання тощо) використовуйте `executeUpdate()` замість `executeQuery()`:

```
int res = stmt.executeUpdate("CREATE TABLE `test`.`a` (`id` INT NOT NULL  
AUTO_INCREMENT, PRIMARY KEY (`id`));");
```

Зверніть увагу на тип результату (`int`, у випадкові використання `executeQuery()` - `ResultSet`).