ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМ. Р.Е. АЛЕКСЕЕВА»

Институт радиоэлектроники и информационных технологий
Кафедра “Прикладная математика”

Отчет по лабораторной работе курса “Базы данных”

Лабораторная работа №2.3
«Взаимодействие с SQL + Java DAO»

Выполнил студент группы 18-ПМ:
Винокуров М.С
Проверил:
Моисеев А.Е.

НИЖНИЙ НОВГОРОД

2021 г.

Оглавление

Введение

DAO

В программном обеспечении data access object (DAO) — абстрактный интерфейс к какому-либо типу базы данных или механизму хранения. Определённые возможности предоставляются независимо от того, какой механизм хранения используется и без необходимости специальным образом соответствовать этому механизму хранения. Этот шаблон проектирования применим ко множеству языков программирования, большинству программного обеспечения, нуждающемуся в хранении информации и к большей части баз данных, но традиционно этот шаблон связывают с приложениями на платформе Java Enterprise Edition, взаимодействующими с реляционными базами данных через интерфейс JDBC, потому что он появился в рекомендациях от фирмы Sun Microsystems.

DAO абстрагирует сущности системы и делает их отображение на БД, определяет общие методы использования соединения, его получение и закрытие.

Вершиной иерархии DAO является абстрактный класс или интерфейс с описанием общих методов, которые будут использоваться при взаимодействии с базой данных. Как правило, это методы поиска, удаление по ключу, обновление и т.д.

SQLite

SQLite— компактная встраиваемая СУБД. Исходный код библиотеки передан в общественное достояние. В 2005 году проект получил награду Google-O'Reilly Open Source Awards.

Слово «встраиваемый» (embedded) означает, что SQLite не использует парадигму клиент-сервер, то есть движок SQLite не является отдельно работающим процессом, с которым взаимодействует программа, а представляет собой библиотеку, с которой программа компонуется, и движок становится составной частью программы. Таким образом, в качестве протокола обмена используются вызовы функций (API) библиотеки SQLite. Такой подход уменьшает накладные расходы, время отклика и упрощает программу. SQLite хранит всю базу данных (включая определения, таблицы, индексы и данные) в единственном стандартном файле на том компьютере, на котором исполняется программа. Простота реализации достигается за счёт того, что перед началом исполнения транзакции записи весь файл, хранящий базу данных, блокируется; ACID-функции достигаются в том числе за счёт создания файла журнала.

Несколько процессов или потоков могут одновременно без каких-либо проблем читать данные из одной базы. Запись в базу можно осуществить только в том случае, если никаких других запросов в данный момент не обслуживается; в противном случае попытка записи оканчивается неудачей, и в программу возвращается код ошибки. Другим вариантом развития событий является автоматическое повторение попыток записи в течение заданного интервала времени. Сама библиотека SQLite написана на C; существует большое количество привязок к другим языкам программирования, в том числе Apple Swift, Delphi, C++, Java, C#, VB.NET, Python, Perl, Node.js, PHP, PureBasic, Tcl (средства для работы с Tcl включены в комплект поставки SQLite), Ruby, Haskell, Scheme, Smalltalk, Lua и Parser, а также ко многим другим. Полный список существующих средств размещён на странице проекта.

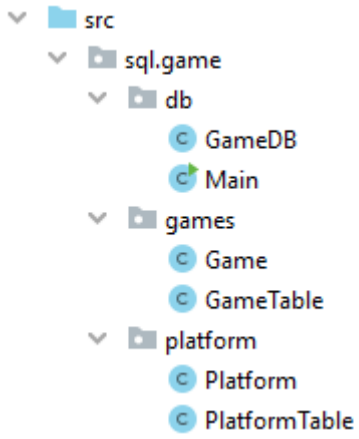
Для NodeJS драйвер SQLite называется *sqlite3* и может быть установлен с помощью *npm.*

Задание

- Перевести код из Лаб. Работы 1 в новый проект, применяя API DAO
- Добавление элементов: Создание, вызов метода класса-таблицы
- Выборка элементов: Вызов метода класса таблицы, возвращающей список
- Добавить дополнительную таблицу, выполнить запрос к связанным таблицам

Выполнение работы

Был создан проект Java, в среде Idea:



В качестве встраиваемой базы данных была выбрана SQLite.
В качестве объектов хранящихся в базе данных выступает список игр, который будет взаимодействовать с платформой, на которой выпущена игра.

Классы будут располагается в независимых файлах, для удобства.
Game.java и Platform.java

```
package sql.game.games;

public class Game{
    public String title, store;
    public int release_date, price, platform_id;
}
```

```
package sql.game.platform;

public class Platform{
    public String device, owner;
    public int platform_id;
}
```

platform_id – связывающий ключ для обеих таблиц.

Теперь, непосредственно, нужно применить интерфейс DAO. Программа работает с двумя независимыми таблицами, описанных через данный API. В моем случае производится работа с абстрактной библиотекой игр, и создаются классы с платформой этих игры и соответственно класс этих игр.

Рассмотрим каждый класс параллельно:

```
public class PlatformTable {
    private ResultSet result;
    private Statement st;
    private PreparedStatement insertSt;
```

PlatformTable.java

```
public class GameTable {
    private ResultSet result;
    private Statement st;
    private PreparedStatement insertSt;
    private PreparedStatement selPlatformSt;
    private PreparedStatement selPriceRangeSt;
```

GameTable.java

ResultSet – содержит результат запроса;
Statement – выполнение запроса;
PreparedStatement *** – предварительные переменные с запросами.

Конструкторы, принимающие аргумент на соединение с базой данных

```
public PlatformTable(Connection connect) throws SQLException {
    result = null;
    insertSt = connect.prepareStatement("INSERT INTO PlatformTable (device, owner) VALUES(?,
?);");
    st = connect.createStatement();
```

PlatformTable.java

```
public GameTable(Connection connect) throws SQLException {
    result = null;
    insertSt = connect.prepareStatement("INSERT INTO GameTable VALUES(?, ?, ?, ?, ?);");
    selPlatformSt = connect.prepareStatement("SELECT * FROM GameTable WHERE platform_id=?");
    selPriceRangeSt = connect.prepareStatement("SELECT * FROM GameTable WHERE (price>=? AND
price<=?);");
    st = connect.createStatement();
```

GameTable.java

Заполним таблицы с данными платформ и игр, создадим методы.

Добавление элементов в таблицу играми:

```
public void insert(Game s) throws SQLException {
    insertSt.setString(1, s.title);
    insertSt.setInt(2, s.release_date);
    insertSt.setInt(3, s.price);
    insertSt.setInt(4, s.platform_id);
    insertSt.setString(5, s.store);
    insertSt.execute();
}
```

GameTable.java

Добавление элементов в таблицу платформ:

```
public void insert(Platform b) throws SQLException {
    insertSt.setString(1, b.device);
    insertSt.setString(2, b.owner);
    insertSt.execute();
}
```

PlatformTable.java

И, соответственно, применим данные методы для заполнения таблицы.

Метод, заполняющий платформы:

```
public void insertData() throws SQLException {
    Platform temp = new Platform();
    temp.device = "PlayStation";
    temp.owner = "Sony";
    insert(temp);

    temp.device = "Xbox";
    temp.owner = "Microsoft";
    insert(temp);

    temp.device = "PC";
    temp.owner = "-";
    insert(temp);
}
```

PlatformTable.java

Метод, заполняющий таблицу с играми, на примере одной:

```
public void insertData() throws SQLException {
    Game data_bridge = new Game();
    //1
    data_bridge.title = "Fallout";
    data_bridge.platform_id = 3;
    data_bridge.price = 40;
    data_bridge.release_date = 1994;
    data_bridge.store = "BethesdaStore";
    insert(data_bridge);
```

GameTable.java

Теперь для каждой из таблиц опишем методы выдачи информации. Всей или по наложенным фильтрам. Они принимают параметр, обрабатывают его, и выдают соответсвенный результат в основное тело программы по запросу.

В каждом файле сделаем отдельные методы передачи.
Сначала рассмотрим методы в GameTable.java:

```
public List<Game> selectAll() throws SQLException {
        List<Game> list = new ArrayList<Game>();
        result = st.executeQuery("SELECT * FROM GameTable");
        while (result.next()) {
            Game item = new Game();
            item.title = result.getString("title");
            item.platform_id = result.getInt("platform_id");
            item.price = result.getInt("price");
            item.release_date = result.getInt("release_date");
            item.store = result.getString("store");
            list.add(item);
        }
        return list;
  }
```

Выборка всех элементов, SelectAll.

```
  public List<Game> selectPlatform(String type) throws SQLException {
        List<Game> list = new ArrayList<Game>();
        selPlatformSt.setString(1, type);
        result = selPlatformSt.executeQuery();
        while (result.next()) {
            Game item = new Game();
            item.title = result.getString("title");
            item.platform_id = result.getInt("platform_id");
            item.price = result.getInt("price");
            item.release_date = result.getInt("release_date");
            item.store = result.getString("store");
            list.add(item);
        }
        return list;
  }
```

Выборка элементов по платформе.

```
  public List<Game> selectPrice(int from, int to) throws SQLException {
        List<Game> list = new ArrayList<Game>();
        selPriceRangeSt.setInt(1, from);
        selPriceRangeSt.setInt(2, to);
        result = selPriceRangeSt.executeQuery();
        while (result.next()) {
            Game item = new Game();
            item.title = result.getString("title");
            item.platform_id = result.getInt("platform_id");
            item.price = result.getInt("price");
            item.release_date = result.getInt("release_date");
            item.store = result.getString("store");
            list.add(item);
        }
        return list;
    }
}
```

Выборка элементов по цене.

У PlatformTable соответственно будет лишь один метод – вывести все.

```
  public List<Platform> selectAll() throws SQLException {
        List<Platform> list = new ArrayList<Platform>();
        result = st.executeQuery("SELECT * FROM PlatformTable");
        while (result.next()) {
            Platform item = new Platform();
            item.device = result.getString("device");
            item.owner = result.getString("owner");
            item.platform_id = result.getInt("platform_id");
            list.add(item);
        }
        return list;
    }
}
```

Выборка всех элементов, SelectAll.

Рассмотрит заключающий класс Main, в котором вызываются все основные методы и производится вывод в консоль, с передачей соответствующих параметров:

```
public static void main(String[] args) throws SQLException{
    GameDB base = new GameDB();
    base.connect();
    PlatformTable b_tab = base.getPlatformTable();
    GameTable s_tab = base.getGameTable();
    System.out.println(" 3 = PC, 2 = Xbox, 1 = PS\n");
    //вывод всего
    for(Game item : s_tab.selectAll()){
        System.out.println(" "+item.platform_id +" "+ item.title+"  "+item.price+"
"+item.release_date);
    }
    System.out.println();

    //вывод по платформе
    for(Game item : s_tab.selectPlatform("2")){
        System.out.println("("+item.platform_id+") "+item.title+"  "+item.price+"
"+item.release_date);
    }
    System.out.println();
    //вывод по деньгам
    for(Game item : s_tab.selectPrice(60, 150)){
        System.out.println("("+item.platform_id+") "+item.title+"  "+item.price+"
"+item.release_date);
    }

    System.out.println();
    base.joinSelect();

}
```

Результаты работы программы

Вывод всех данных

```
Fallout | Price 40$ | Release date: 1994 | PC | BethesdaStore |
Fallout 2 | Price 50$ | Release date: 1996 | PC | BethesdaStore |
Fallout Nevada | Price 0$ | Release date: 2001 | PC | non |
Fallout New Vegas | Price 40$ | Release date: 2009 | PC | Steam |
Halo Combat Evolved | Price 60$ | Release date: 2003 | Xbox | Xbox Live |
Halo 2 | Price 60$ | Release date: 2005 | Xbox | Xbox Live |
Dota 2 | Price 0$ | Release date: 2011 | PC | Steam |
League of legends | Price 0$ | Release date: 2009 | PC | RIOT Launcher |
Counter-Strike Condition Zero | Price 25$ | Release date: 2004 | PC | Steam |
Call of Duty 3 | Price 45$ | Release date: 2005 | Xbox | Xbox Live |
Horizon Zero Dawn | Price 60$ | Release date: 2017 | PlayStation | PS Network |
The last of Us | Price 60$ | Release date: 2013 | PlayStation | PS Network |
Fallout 76 | Price 50$ | Release date: 2018 | PC | BethesdaStore |
Halo 5 | Price 60$ | Release date: 2015 | Xbox | Xbox Live |
Stubbs the Zombie | Price 45$ | Release date: 2004 | Xbox | Xbox Live |
Battlefield 1942 | Price 55$ | Release date: 2002 | PC | Retail Only |
Team Fortress 2 | Price 5$ | Release date: 2008 | PC | Steam |
Half-life | Price 45$ | Release date: 1999 | PC | Retail Only |
Half-Life 2 | Price 60$ | Release date: 2004 | PC | Steam |
Half-Life 3 | Price 120$ | Release date: 2077 | PC | CyberPunkedSteam |
```

Вывод данных по запросу «Steam»

```
x86/x65 Steam  Fallout New Vegas  40$  2009
x86/x65 Steam  Dota 2  0$  2011
x86/x65 Steam  Counter-Strike Condition Zero  25$  2004
x86/x65 Steam  Team Fortress 2  5$  2008
x86/x65 Steam  Half-Life 2  60$  2004
```

Вывод данных по владельцу платформы:

```
Sony PS Network  Horizon Zero Dawn  60$  2017
Sony PS Network  The last of Us  60$  2013
```

Вывод

В ходе выполнения работы был создана база данных, которая работает с интерфейсом DAO. Были реализованы две разные таблицы, которые могли сопоставлятся друг-с-другом, посредством связанного ключа, в нашем случае айди платформы.

Дополнительно была проведена защита передачи запросов при помощи «PreparedStatement», предотвращающая «sql injection»

Список источников

1. Статья о SQLite в Википедии - https://ru.wikipedia.org/wiki/SQLite
2. Статья DAO в Википедии - https://ru.wikipedia.org/wiki/Data_Access_Object
3. Документация NodeJS - https://ru.wikipedia.org/wiki/SQLite

Приложение.
Листинг.

Main.js

```
package sql.game.db;


import java.sql.SQLException;
import sql.game.games.*;
import sql.game.platform.*;

public class Main {

   public static void main(String[] args) throws SQLException{
      GameDB base = new GameDB();
      base.connect();
      PlatformTable b_tab = base.getPlatformTable();
      GameTable s_tab = base.getGameTable();
      System.out.println(" 3 = PC, 2 = Xbox, 1 = PS\n");
      //вывод всего
      for(Game item : s_tab.selectAll()){
         System.out.println(" "+item.platform_id +" "+ item.title+"  "+item.price+"  "+item.release_date);
      }
      System.out.println();

      //вывод по платформе
      for(Game item : s_tab.selectPlatform("2")){
         System.out.println("("+item.platform_id+") "+item.title+"  "+item.price+"  "+item.release_date);
      }
      System.out.println();
      //вывод по деньгам
      for(Game item : s_tab.selectPrice(60, 150)){
         System.out.println("("+item.platform_id+") "+item.title+"  "+item.price+"  "+item.release_date);
      }

      System.out.println();
      base.joinSelect();

   }

}
```

GameDB.js

```
package sql.game.db;
import java.sql.*;
import sql.game.games.GameTable;
import sql.game.platform.PlatformTable;

public class GameDB {
   private Connection connect;
   private Statement st;

   public GameDB() {
      st = null;
      connect= null;
   }

   public void connect() throws SQLException {
      try {
         Class.forName("org.sqlite.JDBC");
      } catch (ClassNotFoundException e) {
         e.printStackTrace();
      }
      connect("jdbc:sqlite:C:\\Users\\super\\Desktop\\Labs\\GameDB-2-3\\db\\GameDB.db");
   }

   public void connect(String filename) throws SQLException {
      connect= DriverManager.getConnection(filename);
   }

   public GameTable getGameTable() throws SQLException{
      GameTable table = null;
      st = connect.createStatement();
      ResultSet result = st.executeQuery("SELECT name FROM sqlite_master WHERE type='table' AND
name='GameTable';");
      if(!result.next()){
         ResultSet result2 = st.executeQuery("SELECT name FROM sqlite_master WHERE type='table' AND
name='PlatformTable';");
         if(!result2.next()){
            getPlatformTable();
         }
         System.out.println("Create Table GameTable");
         st.execute("CREATE TABLE GameTable (title char(100) NOT NULL, release_date integer NOT
NULL, price integer NOT NULL, platform_id integer, store char(20), FOREIGN KEY (platform_id)
REFERENCES PlatformTable(platform_id));");
         table = new GameTable(connect);
         table.insertData();
      }else{
         table = new GameTable(connect);
      }
      return table;
   }

   public PlatformTable getPlatformTable() throws SQLException{
      PlatformTable table = null;
      st = connect.createStatement();
      ResultSet result = st.executeQuery("SELECT name FROM sqlite_master WHERE type='table' AND
name='PlatformTable';");
      if(!result.next()){
         System.out.println("Create Table PlatformTable");
         st.execute("CREATE TABLE PlatformTable (platform_id integer PRIMARY KEY, device char(20)
NOT NULL, owner char(20) NOT NULL);");
```

```
            table = new PlatformTable(connect);
            table.insertData();
        }else{
            table = new PlatformTable(connect);
        }
        return table;
    }

    public void joinSelect() throws SQLException{
        ResultSet res = null;
        st = connect.createStatement();
        res = st.executeQuery("SELECT * FROM GameTable, PlatformTable WHERE
GameTable.platform_id=PlatformTable.platform_id");
        while (res.next()){
            System.out.println(res.getString("title")+" | Price "+res.getString("price")+"$ |"+" Release date:
"+res.getInt("release_date")+" | "+res.getString("device")+" | "+res.getString("store")+" |" );
        }
    }
}
```

Game.js

```
package sql.game.games;

public class Game{
    public String title, store;
    public int release_date, price, platform_id;
}
```

Platform.js

```
package sql.game.platform;

public class Platform{
    public String device, owner;
    public int platform_id;
}
```

GameTable.js

```
package sql.game.games;

import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class GameTable {
    private ResultSet result;
    private Statement st;
    private PreparedStatement insertSt;
    private PreparedStatement selPlatformSt;
    private PreparedStatement selPriceRangeSt;

    public GameTable(Connection connect) throws SQLException {
        result = null;
        insertSt = connect.prepareStatement("INSERT INTO GameTable VALUES(?, ?, ?, ?, ?);");
        selPlatformSt = connect.prepareStatement("SELECT * FROM GameTable WHERE platform_id=?");
```

```
        selPriceRangeSt = connect.prepareStatement("SELECT * FROM GameTable WHERE (price>=? AND
price<=?);");
        st = connect.createStatement();
    }

    public void insert(Game s) throws SQLException {
        insertSt.setString(1, s.title);
        insertSt.setInt(2, s.release_date);
        insertSt.setInt(3, s.price);
        insertSt.setInt(4, s.platform_id);
        insertSt.setString(5, s.store);
        insertSt.execute();
    }

    public void insertData() throws SQLException {
        Game data_bridge = new Game();
        //1
        data_bridge.title = "Fallout";
        data_bridge.platform_id = 3;
        data_bridge.price = 40;
        data_bridge.release_date = 1994;
        data_bridge.store = "BethesdaStore";
        insert(data_bridge);
        //2
        data_bridge.title = "Fallout 2";
        data_bridge.platform_id = 3;
        data_bridge.price = 50;
        data_bridge.release_date = 1996;
        data_bridge.store = "BethesdaStore";
        insert(data_bridge);
        //3
        data_bridge.title = "Fallout Nevada";
        data_bridge.platform_id = 3;
        data_bridge.price = 0;
        data_bridge.release_date = 2001;
        data_bridge.store = "non";
        insert(data_bridge);
        //4
        data_bridge.title = "Fallout New Vegas";
        data_bridge.platform_id = 3;
        data_bridge.price = 40;
        data_bridge.release_date = 2009;
        data_bridge.store = "Steam";
        insert(data_bridge);
        //5
        data_bridge.title = "Halo Combat Evolved";
        data_bridge.platform_id = 2;
        data_bridge.price = 60;
        data_bridge.release_date = 2003;
        data_bridge.store = "Xbox Live";
        insert(data_bridge);
        //6
        data_bridge.title = "Halo 2";
        data_bridge.platform_id = 2;
        data_bridge.price = 60;
        data_bridge.release_date = 2005;
        data_bridge.store = "Xbox Live";
        insert(data_bridge);
        //7
        data_bridge.title = "Dota 2";
        data_bridge.platform_id = 3;
        data_bridge.price = 0;
```

```
data_bridge.release_date = 2011;
data_bridge.store = "Steam";
insert(data_bridge);
//8
data_bridge.title = "League of legends";
data_bridge.platform_id = 3;
data_bridge.price = 0;
data_bridge.release_date = 2009;
data_bridge.store = "RIOT Launcher";
insert(data_bridge);
//9
data_bridge.title = "Counter-Strike Condition Zero";
data_bridge.platform_id = 3;
data_bridge.price = 25;
data_bridge.release_date = 2004;
data_bridge.store = "Steam";
insert(data_bridge);
//10
data_bridge.title = "Call of Duty 3";
data_bridge.platform_id = 2;
data_bridge.price = 45;
data_bridge.release_date = 2005;
data_bridge.store = "Xbox Live";
insert(data_bridge);
//11
data_bridge.title = "Horizon Zero Dawn";
data_bridge.platform_id = 1;
data_bridge.price = 60;
data_bridge.release_date = 2017;
data_bridge.store = "PS Network";
insert(data_bridge);
//12
data_bridge.title = "The last of Us";
data_bridge.platform_id = 1;
data_bridge.price = 60;
data_bridge.release_date = 2013;
data_bridge.store = "PS Network";
insert(data_bridge);
//13
data_bridge.title = "Fallout 76";
data_bridge.platform_id = 3;
data_bridge.price = 50;
data_bridge.release_date = 2018;
data_bridge.store = "BethesdaStore";
insert(data_bridge);
//14
data_bridge.title = "Halo 5";
data_bridge.platform_id = 2;
data_bridge.price = 60;
data_bridge.release_date = 2015;
data_bridge.store = "Xbox Live";
insert(data_bridge);
//15
data_bridge.title = "Stubbs the Zombie";
data_bridge.platform_id = 2;
data_bridge.price = 45;
data_bridge.release_date = 2004;
data_bridge.store = "Xbox Live";
insert(data_bridge);
//16
data_bridge.title = "Battlefield 1942";
data_bridge.platform_id = 3;
```

```
    data_bridge.price = 55;
    data_bridge.release_date = 2002;
    data_bridge.store = "Retail Only";
    insert(data_bridge);
    //17
    data_bridge.title = "Team Fortress 2";
    data_bridge.platform_id = 3;
    data_bridge.price = 5;
    data_bridge.release_date = 2008;
    data_bridge.store = "Steam";
    insert(data_bridge);
    //18
    data_bridge.title = "Half-life";
    data_bridge.platform_id = 3;
    data_bridge.price = 45;
    data_bridge.release_date = 1999;
    data_bridge.store = "Retail Only";
    insert(data_bridge);
    //19
    data_bridge.title = "Half-Life 2";
    data_bridge.platform_id = 3;
    data_bridge.price = 60;
    data_bridge.release_date = 2004;
    data_bridge.store = "Steam";
    insert(data_bridge);
    //20
    data_bridge.title = "Half-Life 3";
    data_bridge.platform_id = 3;
    data_bridge.price = 120;
    data_bridge.release_date = 2077;
    data_bridge.store = "CyberPunkedSteam";
    insert(data_bridge);

}

public List<Game> selectAll() throws SQLException {
    List<Game> list = new ArrayList<Game>();
    result = st.executeQuery("SELECT * FROM GameTable");
    while (result.next()) {
        Game item = new Game();
        item.title = result.getString("title");
        item.platform_id = result.getInt("platform_id");
        item.price = result.getInt("price");
        item.release_date = result.getInt("release_date");
        item.store = result.getString("store");
        list.add(item);
    }
    return list;
}

public List<Game> selectPlatform(String type) throws SQLException {
    List<Game> list = new ArrayList<Game>();
    selPlatformSt.setString(1, type);
    result = selPlatformSt.executeQuery();
    while (result.next()) {
        Game item = new Game();
        item.title = result.getString("title");
        item.platform_id = result.getInt("platform_id");
        item.price = result.getInt("price");
        item.release_date = result.getInt("release_date");
        item.store = result.getString("store");
        list.add(item);
```

```
        }
        return list;
    }

    public List<Game> selectPrice(int from, int to) throws SQLException {
        List<Game> list = new ArrayList<Game>();
        selPriceRangeSt.setInt(1, from);
        selPriceRangeSt.setInt(2, to);
        result = selPriceRangeSt.executeQuery();
        while (result.next()) {
            Game item = new Game();
            item.title = result.getString("title");
            item.platform_id = result.getInt("platform_id");
            item.price = result.getInt("price");
            item.release_date = result.getInt("release_date");
            item.store = result.getString("store");
            list.add(item);
        }
        return list;
    }
}
```

PlatformTable.js

```
package sql.game.platform;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class PlatformTable {
    private ResultSet result;
    private Statement st;
    private PreparedStatement insertSt;

    public PlatformTable(Connection connect) throws SQLException {
        result = null;
        insertSt = connect.prepareStatement("INSERT INTO PlatformTable (device, owner) VALUES(?, ?);");
        st = connect.createStatement();
    }
    public void insert(Platform b) throws SQLException {
        insertSt.setString(1, b.device);
        insertSt.setString(2, b.owner);
        insertSt.execute();
    }
    //platform_id = 1 = PS
    //platform_id = 2 = Xbox
    //platform_id = 3 = PC
    public void insertData() throws SQLException {
        Platform temp = new Platform();
        temp.device = "PlayStation";
        temp.owner = "Sony";
        insert(temp);

        temp.device = "Xbox";
        temp.owner = "Microsoft";
        insert(temp);

        temp.device = "PC";
        temp.owner = "-";
        insert(temp);
```

```
    }

    public List<Platform> selectAll() throws SQLException {
        List<Platform> list = new ArrayList<Platform>();
        result = st.executeQuery("SELECT * FROM PlatformTable");
        while (result.next()) {
            Platform item = new Platform();
            item.device = result.getString("device");
            item.owner = result.getString("owner");
            item.platform_id = result.getInt("platform_id");
            list.add(item);
        }
        return list;
    }
}
```