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 №3
«Отображение ORM»

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

НИЖНИЙ НОВГОРОД

2021 г.

Оглавление

Введение

ORM

ORM (Object-Relational Mapping) – технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных». Существуют как проприетарные, так и свободные реализации этой технологии.

Библиотеки ORM существуют для самых разных языков программирования. В общих чертах, технология ORM позволяет проектировать работу с данными в терминах классов, а не таблиц данных. Она позволяет преобразовывать классы в данные, пригодные для хранения в базе данных, причем схему преобразования определяет сам разработчик. Кроме того, ORM предоставляет простой API- интерфейс для CRUD-операций над данными. Благодаря технологии ORM нет необходимости писать SQL-код для взаимодействия с локальной базой данных.

Среди достоинств ORM выделяют:

· наличие явного описания схемы БД, представленное в терминах какого-либо языка программирования, которое находится и редактируется в одном месте;

· возможность оперировать элементами языка программирования, т.е. классами, объектами, атрибутами, методами, а не элементами реляционной модели данных;

· возможность автоматического создания SQL-запросов, которая избавляет от необходимости использования языка для описания структуры БД (Data Definition Language) и языка манипулирования данными (Data Manipulation Language) при проектировании БД и изменении её схемы соответственно;

· не нужно создавать новые SQL-запросы при переносе на другую систему управления базами данных, поскольку за это отвечает низкоуровневый драйвер ORM.

· ORM избавляет от необходимости работы с SQL и проработки значительного количества программного кода, который зачастую однообразен и подвержен ошибкам.

· код, генерируемый ORM гипотетически проверен и оптимизирован, следовательно не нужно беспокоиться о его тестировании;

· развитые реализации ORM поддерживают отображение наследования и композиции на таблицы;

· ORM дает возможность изолировать код программы от подробностей хранения данных.

Среди недостатков ORM выделяются:

· Дополнительная нагрузка на программиста, которому, в случае использования ORM необходимо изучать этот некий «дополнительный слой»

между программной и базой данных, который к тому же создает дополнительный уровень абстракции — объекты ORM. В связи с этим могут возникнуть вопросы соответствия особенностям ООП и соответствующим реляционным операциям. Эту проблему называют impedance mismatch, а сама реализация ORM ведет к увеличению объема программного кода и снижению скорости работы программы. Однако, с другой стороны, ORM наглядно и в одном месте концентрирует различие между реляционной и объектно-ориентированной парадигмами, что нельзя назвать недостатком;

· Появление трудно поддающихся отладке ошибок в программе, если присутствуют ошибки в реализации ORM, например, ошибки в реализации кэширования ORM, такие как согласование изменений в разных сессиях.

· Недостатки реализаций, которые могут иметь определенные ограничения и выдвигать определенные требования, например, требование собственной схемы базы данных и ограничение на средства создания базы данных. Также может отсутствовать возможность написать в явном виде SQL-запрос.

· Требуются отдельные таблицы в случае прямого отображения классов в таблицы и необходимости отображения атрибутов множественного характера.

Если говорить о главном минусе ORM, снижении производительности, то причина этого состоит в том, что большинство из ORM нацелены на обработку значительного большего количества различных сценариев использования данных, чем в случае отдельного приложения. В случае небольших проектов, которые не сталкиваются с высокой нагрузкой, применение ORM очевидно, особенно, если учесть такой важный критерий разработки, как время. «То, что с легкостью пишется с использованием ORM за неделю, можно реализовывать ни один месяц собственными усилиями».

OrmLite

ORMLite - легковесная библиотека ORM для приложений Java. Он предоставляет стандартные функции инструмента ORM для наиболее распространенных случаев использования, без дополнительной сложности и накладных расходов других структур ORM. Это основные функции:

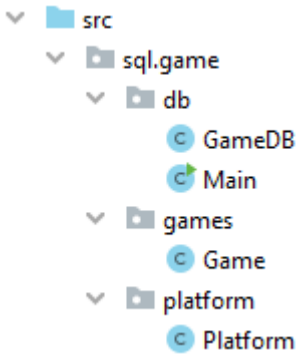
- Определение классов сущностей с использованием JAVA;
- Расширяемые DAO классы
- QurryBuilder для сложных запросов
- Поддержка трнзакций
- Поддержка отношений сущностей

Задание

- Перевести код из Лаб. Работы 2 в новый проект, применяя ORM.

Выполнение работы

Был конвертирован проект Java с прошлой работы, в среде Idea:



Фреймворк устанавливаем через Maven встроенный в Idea, выберем OrmLite последней версии.
В качестве объектов хранящихся в базе данных выступает старый список игр, который будет взаимодействовать с платформой, на которой выпущена игра. В целом структура всего продукта сохраняется с прошлой выполненной работы на основе DAO.

Классы будут располагается в независимых файлах, для удобства, относительно прошлой работы с DAO, придется видоизменить код для добавления филдов.

Game.java

```
@DatabaseTable(tableName = "GameTableEx")
public class Game{
    @DatabaseField(generatedId = true)
    public int id;

    @DatabaseField(canBeNull = false, dataType = DataType.STRING)
    public String title;

    @DatabaseField(canBeNull = false, dataType = DataType.STRING)
    public String store;

    @DatabaseField(canBeNull = false, dataType = DataType.INTEGER)
    public int release_date;

    @DatabaseField(canBeNull = false, dataType = DataType.INTEGER)
    public int price;

    @DatabaseField(foreign = true, foreignColumnName = "platform_id", foreignAutoRefresh =
true, foreignAutoCreate = true)
    public Platform platform;
}
```

Platform.java

```
@DatabaseTable(tableName = "PlatformTableEx")
public class Platform{
    @DatabaseField(generatedId = true)
    public int platform_id;

    @DatabaseField(canBeNull = false, dataType = DataType.STRING)
    public String device;

    @DatabaseField(canBeNull = false, dataType = DataType.STRING)
    public String owner;

}
```

Используем аннотации, для предоставления фреймворку способ хранения данных в самой БД. Применяем @DatabaseTable для передачи места хранения класса, и @DatabaseField для описания хранения поля класса в таблице.

В GameDB.java объединим методы для получения данных, по-сравнению с прошлой работой. Рассмотрим все методы:

```
public List<Game> selectAll() throws SQLException{
    List<Game> list = new ArrayList<Game>();
    QueryBuilder<Game, Integer> GameQB = GamesDAO.queryBuilder();
    QueryBuilder<Platform, Integer> PlatformQB = PlatformsDAO.queryBuilder();
    GameQB.join(PlatformQB);
    PreparedQuery<Game> preparedQuery= GameQB.prepare();
    CloseableIterator<Game> it = GamesDAO.iterator(preparedQuery);
    while (it.hasNext()){
        Game data_bridge = it.current();
        System.out.println(data_bridge.platform.owner + "| " + data_bridge.platform.device + "
| " + data_bridge.title + " | " + data_bridge.release_date + " | " + data_bridge.price + " | "
+ data_bridge.store);
        it.moveToNext();
        list.add(data_bridge);
    }
    return list;
}
```

```
public List<Game> selectStore(String store) throws SQLException {
    List<Game> list = new ArrayList<Game>();
    QueryBuilder<Game, Integer> GameQB = GamesDAO.queryBuilder();
    GameQB.where().like("store", store);
    CloseableIterator<Game> it = GamesDAO.iterator(GameQB.prepare());
    while (it.hasNext()){
        Game data_bridge = it.current();
        list.add(data_bridge);
        it.moveToNext();
    }
    return list;
}
```

```
public List<Game> selectOwner(String owner) throws SQLException {
    List<Game> list = new ArrayList<Game>();
    QueryBuilder<Game, Integer> GameQB = GamesDAO.queryBuilder();
    QueryBuilder<Platform, Integer> PlatformQB = PlatformsDAO.queryBuilder();
    PlatformQB.where().like("owner", owner);
    GameQB.join(PlatformQB);
    CloseableIterator<Game> it = GamesDAO.iterator(GameQB.prepare());
    while (it.hasNext()){
        Game data_bridge = it.current();
        list.add(data_bridge);
        it.moveToNext();
    }
    return list;
}


public List<Game> selectPrice(int from, int to) throws SQLException {
    List<Game> list = new ArrayList<Game>();
    QueryBuilder<Game, Integer> GameQB = GamesDAO.queryBuilder();
    GameQB.where().between("price", from, to);
    CloseableIterator<Game> it = GamesDAO.iterator(GameQB.prepare());
    while (it.hasNext()){
        Game data_bridge = it.current();
        list.add(data_bridge);
        it.moveToNext();
    }
    return list;
}
```

А так же видоизменим инсерты, для добавления новых данных в базу данных. Сначала опишем вид добавления данных о платформе игр, а далее создадим один ввод игры в базу данных

```
private void insertGameData() throws SQLException {
    Platform Platform1 = new Platform();
    Platform Platform2 = new Platform();
    Platform Platform3 = new Platform();
    Platform1.device = "PS";
    Platform1.owner = "Sony";
    Platform2.device = "Xbox";
    Platform2.owner = "Microsoft";
    Platform3.device = "PC";
    Platform3.owner = "x86/x65";

    Game data_bridge = new Game();

    //1
    data_bridge.title = "Fallout";
    data_bridge.platform = Platform3;
    data_bridge.price = 40;
    data_bridge.release_date = 1994;
    data_bridge.store = "BethesdaStore";
    GamesDAO.create(data_bridge);
```

В классе Main опишем вызов функции подключения к базе данных и то, что нам нужно вывести в консоль. Фильтруем требуемые нам данные при помощи ранее описанных селектов.

Main.java

```
public class Main {

    public static void main(String[] args) throws SQLException {
        GameDB db= new GameDB();
        db.connect();
        db.selectAll();

        System.out.println(" ");
        for(Game item : db.selectStore("Steam")){
            System.out.println(item.platform.owner+" "+item.store+"  "+item.title+"
"+item.price+"$  "+item.release_date);
        }
        /*System.out.println(" ");
        for(Game item : db.selectPrice(60, 120)){
            System.out.println(item.platform.owner+" "+item.store+"  "+item.title+"
"+item.price+"$  "+item.release_date);
        }*/
        System.out.println(" ");
        for(Game item : db.selectOwner("Sony")){
            System.out.println(item.platform.owner+" "+item.store+"  "+item.title+"
"+item.price+"$  "+item.release_date);

        }
    }
}
```

Результаты работы программы

Вывод всех данных

```
x86/x65| PC | Fallout | 1994 | 40 | BethesdaStore
x86/x65| PC | Fallout 2 | 1996 | 50 | BethesdaStore
x86/x65| PC | Fallout Nevada | 2001 | 0 | Open-source
x86/x65| PC | Fallout New Vegas | 2009 | 40 | Steam
Microsoft| Xbox | Halo Combat Evolved | 2003 | 60 | Xbox Live
Microsoft| Xbox | Halo 2 | 2005 | 60 | Xbox Live
x86/x65| PC | Dota 2 | 2011 | 0 | Steam
x86/x65| PC | League of legends | 2009 | 0 | RIOT Launcher
x86/x65| PC | Counter-Strike Condition Zero | 2004 | 25 | Steam
Microsoft| Xbox | Call of Duty 3 | 2005 | 45 | Xbox Live
Sony| PS | Horizon Zero Dawn | 2017 | 60 | PS Network
Sony| PS | The last of Us | 2013 | 60 | PS Network
x86/x65| PC | Fallout 76 | 2018 | 50 | BethesdaStore
Microsoft| Xbox | Halo 5 | 2015 | 60 | Xbox Live
Microsoft| Xbox | Stubbs the Zombie | 2004 | 45 | Xbox Live
x86/x65| PC | Battlefield 1942 | 2002 | 55 | Retail Only
x86/x65| PC | Team Fortress 2 | 2008 | 5 | Steam
x86/x65| PC | Half-life | 1999 | 45 | Retail Only
x86/x65| PC | Half-Life 2 | 2004 | 60 | Steam
x86/x65| PC | Half-Life 3 | 2077 | 120 | CyberPunkedSteam
```

Вывод данных по запросу «Steam»

```
Microsoft Xbox Live  Halo Combat Evolved  60$  2003
Microsoft Xbox Live  Halo 2  60$  2005
Microsoft Xbox Live  Call of Duty 3  45$  2005
Microsoft Xbox Live  Halo 5  60$  2015
Microsoft Xbox Live  Stubbs the Zombie  45$  2004
```

Вывод данных от 60 до 120:

```
Microsoft Xbox Live  Halo Combat Evolved  60$  2003
Microsoft Xbox Live  Halo 2  60$  2005
Sony PS Network  Horizon Zero Dawn  60$  2017
Sony PS Network  The last of Us  60$  2013
Microsoft Xbox Live  Halo 5  60$  2015
x86/x65 Steam  Half-Life 2  60$  2004
x86/x65 CyberPunkedSteam  Half-Life 3  120$  2077
```

Вывод данных с владельцем «Майкрософт»

```
Microsoft Xbox Live  Halo Combat Evolved  60$  2003
Microsoft Xbox Live  Halo 2  60$  2005
Microsoft Xbox Live  Call of Duty 3  45$  2005
Microsoft Xbox Live  Halo 5  60$  2015
Microsoft Xbox Live  Stubbs the Zombie  45$  2004
```

Вывод

В ходе выполнения работы был использован ORM, при помощи фреймворка OrmLite. Сделаны связанные запросы через QuerryBuilder.

Удалось уменьшить количество кода, а так же уйти от лишних затрат времени на подробное описание методов запроса/записи/вывода

Список источников

1. ORM в Android с помощью ORMLite - https://habr.com/ru/post/143431/
2. Основная страница OrmLite - http://ormlite.com/
3. Статья о ORM - https://ru.bmstu.wiki/ORM_(Object-Relational_Mapping)
4. Статья об использовании OrmLite - https://www.codeflow.site/ru/article/ormlite

Приложение.
Листинг.

Main.js

```
package sql.game.db;

import sql.game.games.Game;
import java.sql.SQLException;

public class Main {

   public static void main(String[] args) throws SQLException {
      GameDB db= new GameDB();
      db.connect();
      db.selectAll();

      System.out.println(" ");
      for(Game item : db.selectStore("Xbox Live")){
         System.out.println(item.platform.owner+" "+item.store+"  "+item.title+"  "+item.price+"$
"+item.release_date);
      }
      System.out.println(" ");
      for(Game item : db.selectPrice(60, 121)){
         System.out.println(item.platform.owner+" "+item.store+"  "+item.title+"  "+item.price+"$
"+item.release_date);
      }
      System.out.println(" ");
      for(Game item : db.selectOwner("Microsoft")){
         System.out.println(item.platform.owner+" "+item.store+"  "+item.title+"  "+item.price+"$
"+item.release_date);

      }
   }
}
```

GameDB.js

```
package sql.game.db;

import java.sql.*;
import java.util.ArrayList;
import java.util.List;
import sql.game.games.Game;
import sql.game.platform.Platform;

import com.j256.ormlite.dao.CloseableIterator;
import com.j256.ormlite.dao.Dao;
import com.j256.ormlite.dao.DaoManager;
import com.j256.ormlite.jdbc.JdbcConnectionSource;
import com.j256.ormlite.logger.LocalLog;
import com.j256.ormlite.stmt.PreparedQuery;
import com.j256.ormlite.stmt.QueryBuilder;
import com.j256.ormlite.support.ConnectionSource;
import com.j256.ormlite.table.TableUtils;

public class GameDB {

  private ConnectionSource connectionSource;
  private Dao<Game, Integer> GamesDAO;
  private Dao<Platform, Integer> PlatformsDAO;

  public GameDB() throws SQLException {
    try {
      //Loading the sqlite drivers
      Class.forName("org.sqlite.JDBC");
    } catch (ClassNotFoundException e) {
      //Should never happen
      throw new SQLException(e);
    }
    System.setProperty(LocalLog.LOCAL_LOG_FILE_PROPERTY, "db.log");
  }

  public void connect() throws SQLException {
    connect("jdbc:sqlite:.\\db\\GameDB.db");
  }

  public void connect(String filename) throws SQLException {
    connectionSource = new JdbcConnectionSource(filename);

    PlatformsDAO = DaoManager.createDao(connectionSource, Platform.class);
    TableUtils.createTableIfNotExists(connectionSource, Platform.class);
    GamesDAO = DaoManager.createDao(connectionSource, Game.class);
    if(!GamesDAO.isTableExists()) {
      TableUtils.createTable(connectionSource, Game.class);
      insertGameData();
    }
  }

  public List<Game> selectAll() throws SQLException{
    List<Game> list = new ArrayList<Game>();
    QueryBuilder<Game, Integer> GameQB = GamesDAO.queryBuilder();
    QueryBuilder<Platform, Integer> PlatformQB = PlatformsDAO.queryBuilder();
    GameQB.join(PlatformQB);
    PreparedQuery<Game> preparedQuery= GameQB.prepare();
    CloseableIterator<Game> it = GamesDAO.iterator(preparedQuery);
    while (it.hasNext()){
```

```
            Game data_bridge = it.current();
            System.out.println(data_bridge.platform.owner + "| " + data_bridge.platform.device + " | " +
data_bridge.title + " | " + data_bridge.release_date + " | " + data_bridge.price + " | " + data_bridge.store);
            it.moveToNext();
            list.add(data_bridge);
        }
        return list;
    }

    public List<Game> selectStore(String store) throws SQLException {
        List<Game> list = new ArrayList<Game>();
        QueryBuilder<Game, Integer> GameQB = GamesDAO.queryBuilder();
        GameQB.where().like("store", store);
        CloseableIterator<Game> it = GamesDAO.iterator(GameQB.prepare());
        while (it.hasNext()){
            Game data_bridge = it.current();
            list.add(data_bridge);
            it.moveToNext();
        }
        return list;
    }

    public List<Game> selectOwner(String owner) throws SQLException {
        List<Game> list = new ArrayList<Game>();
        QueryBuilder<Game, Integer> GameQB = GamesDAO.queryBuilder();
        QueryBuilder<Platform, Integer> PlatformQB = PlatformsDAO.queryBuilder();
        PlatformQB.where().like("owner", owner);
        GameQB.join(PlatformQB);
        CloseableIterator<Game> it = GamesDAO.iterator(GameQB.prepare());
        while (it.hasNext()){
            Game data_bridge = it.current();
            list.add(data_bridge);
            it.moveToNext();
        }
        return list;
    }

    public List<Game> selectPrice(int from, int to) throws SQLException {
        List<Game> list = new ArrayList<Game>();
        QueryBuilder<Game, Integer> GameQB = GamesDAO.queryBuilder();
        GameQB.where().between("price", from, to);
        CloseableIterator<Game> it = GamesDAO.iterator(GameQB.prepare());
        while (it.hasNext()){
            Game data_bridge = it.current();
            list.add(data_bridge);
            it.moveToNext();
        }
        return list;
    }

    private void insertGameData() throws SQLException {
        Platform Platform1 = new Platform();
        Platform Platform2 = new Platform();
        Platform Platform3 = new Platform();
        Platform1.device = "PS";
        Platform1.owner = "Sony";
        Platform2.device = "Xbox";
        Platform2.owner = "Microsoft";
        Platform3.device = "PC";
        Platform3.owner = "x86/x65";

        Game data_bridge = new Game();
```

```
//1
data_bridge.title = "Fallout";
data_bridge.platform = Platform3;
data_bridge.price = 40;
data_bridge.release_date = 1994;
data_bridge.store = "BethesdaStore";
GamesDAO.create(data_bridge);
//2
data_bridge.title = "Fallout 2";
data_bridge.platform = Platform3;
data_bridge.price = 50;
data_bridge.release_date = 1996;
data_bridge.store = "BethesdaStore";
GamesDAO.create(data_bridge);
//3
data_bridge.title = "Fallout Nevada";
data_bridge.platform = Platform3;
data_bridge.price = 0;
data_bridge.release_date = 2001;
data_bridge.store = "Open-source";
GamesDAO.create(data_bridge);
//4
data_bridge.title = "Fallout New Vegas";
data_bridge.platform = Platform3;
data_bridge.price = 40;
data_bridge.release_date = 2009;
data_bridge.store = "Steam";
GamesDAO.create(data_bridge);
//5
data_bridge.title = "Halo Combat Evolved";
data_bridge.platform = Platform2;
data_bridge.price = 60;
data_bridge.release_date = 2003;
data_bridge.store = "Xbox Live";
GamesDAO.create(data_bridge);
//6
data_bridge.title = "Halo 2";
data_bridge.platform = Platform2;
data_bridge.price = 60;
data_bridge.release_date = 2005;
data_bridge.store = "Xbox Live";
GamesDAO.create(data_bridge);
//7
data_bridge.title = "Dota 2";
data_bridge.platform = Platform3;
data_bridge.price = 0;
data_bridge.release_date = 2011;
data_bridge.store = "Steam";
GamesDAO.create(data_bridge);
//8
data_bridge.title = "League of legends";
data_bridge.platform = Platform3;
data_bridge.price = 0;
data_bridge.release_date = 2009;
data_bridge.store = "RIOT Launcher";
GamesDAO.create(data_bridge);
//9
data_bridge.title = "Counter-Strike Condition Zero";
data_bridge.platform = Platform3;
data_bridge.price = 25;
data_bridge.release_date = 2004;
```

```
data_bridge.store = "Steam";
GamesDAO.create(data_bridge);
//10
data_bridge.title = "Call of Duty 3";
data_bridge.platform = Platform2;
data_bridge.price = 45;
data_bridge.release_date = 2005;
data_bridge.store = "Xbox Live";
GamesDAO.create(data_bridge);
//11
data_bridge.title = "Horizon Zero Dawn";
data_bridge.platform = Platform1;
data_bridge.price = 60;
data_bridge.release_date = 2017;
data_bridge.store = "PS Network";
GamesDAO.create(data_bridge);
//12
data_bridge.title = "The last of Us";
data_bridge.platform = Platform1;
data_bridge.price = 60;
data_bridge.release_date = 2013;
data_bridge.store = "PS Network";
GamesDAO.create(data_bridge);
//13
data_bridge.title = "Fallout 76";
data_bridge.platform = Platform3;
data_bridge.price = 50;
data_bridge.release_date = 2018;
data_bridge.store = "BethesdaStore";
GamesDAO.create(data_bridge);
//14
data_bridge.title = "Halo 5";
data_bridge.platform = Platform2;
data_bridge.price = 60;
data_bridge.release_date = 2015;
data_bridge.store = "Xbox Live";
GamesDAO.create(data_bridge);
//15
data_bridge.title = "Stubbs the Zombie";
data_bridge.platform = Platform2;
data_bridge.price = 45;
data_bridge.release_date = 2004;
data_bridge.store = "Xbox Live";
GamesDAO.create(data_bridge);
//16
data_bridge.title = "Battlefield 1942";
data_bridge.platform = Platform3;
data_bridge.price = 55;
data_bridge.release_date = 2002;
data_bridge.store = "Retail Only";
GamesDAO.create(data_bridge);
//17
data_bridge.title = "Team Fortress 2";
data_bridge.platform = Platform3;
data_bridge.price = 5;
data_bridge.release_date = 2008;
data_bridge.store = "Steam";
GamesDAO.create(data_bridge);
//18
data_bridge.title = "Half-life";
data_bridge.platform = Platform3;
data_bridge.price = 45;
```

```
        data_bridge.release_date = 1999;
        data_bridge.store = "Retail Only";
        GamesDAO.create(data_bridge);
        //19
        data_bridge.title = "Half-Life 2";
        data_bridge.platform = Platform3;
        data_bridge.price = 60;
        data_bridge.release_date = 2004;
        data_bridge.store = "Steam";
        GamesDAO.create(data_bridge);
        //20
        data_bridge.title = "Half-Life 3";
        data_bridge.platform = Platform3;
        data_bridge.price = 120;
        data_bridge.release_date = 2077;
        data_bridge.store = "CyberPunkedSteam";
        GamesDAO.create(data_bridge);
    }

}
```

Game.js

```
package sql.game.games;

import com.j256.ormlite.field.DataType;
import com.j256.ormlite.field.DatabaseField;
import com.j256.ormlite.table.DatabaseTable;
import sql.game.platform.Platform;

@DatabaseTable(tableName = "GameTableEx")
public class Game{
    @DatabaseField(generatedId = true)
    public int id;

    @DatabaseField(canBeNull = false, dataType = DataType.STRING)
    public String title;

    @DatabaseField(canBeNull = false, dataType = DataType.STRING)
    public String store;

    @DatabaseField(canBeNull = false, dataType = DataType.INTEGER)
    public int release_date;

    @DatabaseField(canBeNull = false, dataType = DataType.INTEGER)
    public int price;

    @DatabaseField(foreign = true, foreignColumnName = "platform_id", foreignAutoRefresh = true,
foreignAutoCreate = true)
    public Platform platform;
}
```

Platform.js

```
package sql.game.platform;

import com.j256.ormlite.field.DataType;
import com.j256.ormlite.field.DatabaseField;
import com.j256.ormlite.table.DatabaseTable;

@DatabaseTable(tableName = "PlatformTableEx")
public class Platform{
   @DatabaseField(generatedId = true)
   public int platform_id;

   @DatabaseField(canBeNull = false, dataType = DataType.STRING)
   public String device;

   @DatabaseField(canBeNull = false, dataType = DataType.STRING)
   public String owner;

}
```