Ejemplo de HotRod

Este ejemplo usa el generador MyBatis-Spring que produce código para Java con Spring, MyBatis y Maven. Usa las primitivas CRUD de HotRod para obtener e insertar datos en una base de datos Oracle. Los módulos HotRod LiveSQL y HotRod Nitro no están demostrados en este ejemplo, pero pueden agregados fácilmente.

1. Prepare la base de datos Oracle

El ejemplo necesita de una de base de datos Oracle 10g (o superior) en donde se utiliza un par de tablas a modo de demostración. Para crear las tablas lea el *Anexo 1 - Preparar las tablas de la base de datos*.

2. Cree el proyecto Maven

Ir al directorio padre (dentro de este se creará el proyecto). Maven creará el proyecto en un subdirectorio llamado app1 (de acuerdo al parámetro artifactId). Reemplace los valores de acuerdo a su proyecto y ejecute el comando:

```
mvn archetype:generate -DinteractiveMode=false \
  -DarchetypeGroupId=org.hotrodorm.hotrod \
  -DarchetypeArtifactId=hotrod-archetype-sm-jar-app \
  -DarchetypeVersion=3.2.0 \
  -DgroupId=com.compania                                    \
  -DartifactId=app1                                         \
  -Dversion=1.0.0-SNAPSHOT                                  \
  -Dpackage=com.compania.app1                               \
  -Dpersistencepackage=persistencia                         \
  -Djdbcdriverclassname="oracle.jdbc.driver.OracleDriver" \
  -Djdbcurl="jdbc:oracle:thin:@192.168.56.95:1521:orcl"   \
  -Djdbcusername="user1"                                    \
  -Djdbcpassword="pass1"                                    \
  -Djdbccatalog=""                                          \
  -Djdbcschema=USER1                                        \
  -Djdbcdrivergroupid=com.oracle.ojdbc                      \
  -Djdbcdriverartifactid=ojdbc8                             \
  -Djdbcdriverversion=19.3.0.0                              \
  -Djdbcdrivertype=jar
```

Reemplace los parámetros en rojo para indicar los detalles del nuevo proyecto; los parámetros en azul indican los detalles de la base de datos (tal vez local) en donde están presentes las tablas ya creadas; los parámetros en verde representan los clasificadores Maven del driver JDBC que desee utilizar.

Ahora que el proyecto está creado entre a él:

```
cd app1
```

Aunque el ejemplo puede funcionar enteramente desde línea de comando, usualmente es más fácil usar Eclipse para editarlo y modificarlo. Para importar el proyecto en Eclipse siga las instrucciones del *Anexo 2 - Importar el proyecto en Eclipse*.

3. Genere las clases Java que representan los objetos de la base de datos

Especifique las tablas de base de datos que HotRod va a considerar. En el archivo src/main/hotrod/hotrod.xml agregue las dos líneas resaltadas en amarillo para incluir las tablas CUENTA y TRANSACCION. El archivo resultante debería quedar como:

```
<?xml version="1.0"?>
<!DOCTYPE hotrod SYSTEM "hotrod.dtd">

<hotrod>

  <generators>
    <mybatis-spring>
      <daos package="com.compania.app1.persistencia" />
      <mappers dir="mappers" />
    </mybatis-spring>
  </generators>

  <table name="cuenta" />
  <table name="transaccion" />

</hotrod>
```

Luego use HotRod para generar todas las clases Java y mappers:

```
mvn hotrod:gen
```

HotRod se conecta a la base de datos, extrae los detalles de las tablas, columnas, secuencias y vistas; luego genera las clases Java y comandos SQL correspondientes. A continuación se encuentra los detalles mostrados durante la generación:

```
[INFO] HotRod version 3.2.0 (build 20200602-004009) - Generate
[INFO]
[INFO] Configuration File: src/main/hotrod/hotrod.xml
[INFO] HotRod Database Adapter: Oracle Adapter
[INFO] Database URL: jdbc:oracle:thin:@192.168.56.95:1521:orcl
[INFO] Database Name: Oracle - version 12.1 (Oracle Database 12c Standard Edition Release
12.1.0.2.0 - 64bit Production)
[INFO] JDBC Driver: Oracle JDBC driver - version 19.3 (19.3.0.0.0) - implements JDBC
Specification 4.2
[INFO]
[INFO] Default Database Schema: USER1
[INFO]
[INFO] Generating all facets.
[INFO]
[INFO] Table CUENTA included.
[INFO] Table TRANSACCION included.
[INFO]
[INFO] Total of: 2 tables, 0 views, 0 enums, 0 DAOs, and 0 sequences -- including 0 queries,
and 0 selects.
```

Ahora refresque el proyecto en Eclipse para que cargue las nuevas clases Java generadas.

Para cada tabla incluida HotRod genera cuatro archivos. Por ejemplo, para la tabla CUENTA generó:

- src/main/java/com/compania/app1/persistencia/primitives/CuentaDAO.java: El **DAO** ofrece métodos para cada operación CRUD (insert, update, delete, select y otras).

- src/main/resources/mappers/primitives/primitives-cuenta.xml: El **Mapper** contiene todos los comandos SQL correspondientes a cada método ofrecido por el DAO.
- src/main/java/com/compania/app1/persistencia/primitives/Cuenta.java: El **VO Espejo** incluye propiedades, getters y setters que representan las columnas de la tabla.
- src/main/java/com/compania/app1/persistencia/CuentaVO.java: El **VO Concreto** hereda del VO Espejo y está vacío; su propósito es ser enriquecido por el programador a medida que sea necesario. Este último no es nunca actualizado por HotRod dado que no incluye ningún detalle relacionado a la tabla.

Con excepción de la última clase, los archivos son refrescados cada vez que se usa el generador. El modelo de clases es actualizado sin interferir con las modificaciones propias del VO Concreto que el programador haya hecho manualmente. Esto permite que el modelo Java represente fielmente a la base de datos, aún cuando ésta sufra cambios contínuos en sus tablas, vistas, secuencias, columnas y/o restricciones SQL.

4. Use los DAOs en la aplicación y ejecútela

El proyecto creado incluye la estructura básica para hacer uso de los DAOs. Abra la clase Java ExampleBean.java y agregue las líneas resaltadas en amarillo (los DAOs como propiedades Spring, el cuerpo del método execute() en donde se comunica con al base de datos y los imports necesarios). Con estos cambios la clase debería quedar como se muestra a continuación:

```
package com.compania.app1;

import java.sql.SQLException;
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Transactional;

import com.compania.app1.persistencia.CuentaVO;
import com.compania.app1.persistencia.TransaccionVO;
import com.compania.app1.persistencia.primitives.CuentaDAO;
import com.compania.app1.persistencia.primitives.TransaccionDAO;

@Component("exampleBean")
public class ExampleBean {

  @Autowired
  private CuentaDAO cuentaDAO;

  @Autowired
  private TransaccionDAO transaccionDAO;

  @Transactional
  public void execute() throws SQLException {

    // 1. Select by PK

    CuentaVO c = this.cuentaDAO.selectByPK(102L);
    System.out.printf(" - Saldo cuenta %s: $%,.2f%n", c.getNombre(), c.getSaldo());

    // 2. Select by Example

    TransaccionVO example = new TransaccionVO();
    example.setIdCuenta(c.getId());
    List<TransaccionVO> transacciones = this.transaccionDAO.selectByExample(example);
    for (TransaccionVO t : transacciones) {
      System.out.printf("   - Transaccion #%d - Fecha %2$td-%2$tb-%2$tY - Monto: $%3$,.2f%n",
        t.getId(), t.getFecha(), t.getMonto());
    }

  }

}
```

Compile la aplicación:

```
mvn compile
```

Y luego ejecute el ejemplo:

```
mvn exec:java -Dexec.cleanupDaemonThreads=false -Dexec.classpathScope=test \
  -Dexec.mainClass="com.compania.app1.App"
```

La aplicación se ejecuta y muestra la información obtenida desde la base de datos:

```
[ Starting example ]
 - Saldo cuenta CC-102: $500.00
   - Transaccion #5110 - Fecha 28-Jan-2020 - Monto: $100,000.00
   - Transaccion #5111 - Fecha 22-Mar-2020 - Monto: $-99,500.00
[ Example complete ]
```

Este ejemplo se conectó a la base de datos y ejecutó dos operaciones CRUD:

- selectByPK() en la tabla CUENTA.
- selectByExample() en la tabla TRANSACCION.

Con esto se da por finalizado el ejemplo. Los DAOs ofrecen una serie de métodos disponibles en cada uno de ellos para realizar las operaciones básicas en una tabla.

Anexo 1 - Preparar las tablas de la base de datos

Ejecute el siguiente script de SQL en una base de datos Oracle:

```
create table cuenta (
  id number(18) not null primary key,
  nombre varchar2(12) not null,
  creada_en date not null,
  estado char(1) not null check (estado in ('R', 'A', 'S', 'C')),
  saldo number(14, 2) not null
);

insert into cuenta (id, nombre, creada_en, estado, saldo)
  values (101, 'CC-101', date '2019-12-03', 'A', 150000);
insert into cuenta (id, nombre, creada_en, estado, saldo)
  values (102, 'CC-102', date '2020-01-15', 'S', 500);
insert into cuenta (id, nombre, creada_en, estado, saldo)
  values (103, 'CC-103', date '2020-04-27', 'A', 72000);

create table transaccion (
  id number(18) not null primary key,
  id_cuenta number(18) not null,
  monto number(14, 2) not null,
  fecha date,
  constraint fk1 foreign key (id_cuenta) references cuenta (id)
);

create index ix1 on transaccion (id_cuenta);

insert into transaccion (id, id_cuenta, monto, fecha)
  values (5100, 101, 100000, date '2019-12-03');
insert into transaccion (id, id_cuenta, monto, fecha)
  values (5101, 101,  70000, date '2020-01-17');
insert into transaccion (id, id_cuenta, monto, fecha)
  values (5102, 101, -20000, date '2020-02-05');

insert into transaccion (id, id_cuenta, monto, fecha)
  values (5110, 102, 100000, date '2020-01-28');
insert into transaccion (id, id_cuenta, monto, fecha)
  values (5111, 102, -99500, date '2020-03-22');

insert into transaccion (id, id_cuenta, monto, fecha)
  values (5120, 103,  72000, date '2020-04-27');
```

Este script incluye la creación de dos tablas e inserta algunas filas con datos en cada una.

Anexo 2 - Importar el proyecto en Eclipse

Para importar el Proyecto en Eclipse siga los pasos siguientes:

```
> File > Import... > Existing Maven Projects [Next]
> [Browse] y encontrar el directorio "app1" [OK]
> [Finish]
```

El proyecto tiene todas las características de Maven incluyendo la configuracion de Java 8 y todas las dependencias necesarias.

Anexo 3 - Resumen de funcionalidades de HotRod

HotRod actualmente soporta 10 bases de datos relacionales: Oracle, DB2 LUW, PostgreSQL, SQL Server, MariaDB, SAP/Sybase ASE, MySQL, H2, HyperSQL y Apache Derby.

Incluye reglas por defecto para determinar el tipo de datos Java de cada columna. Estas reglas pueden ser modificadas agregando reglas OGNL en el archivo hotrod.xml (tag <type-solver>); estas reglas usan las propiedades de la columna para determinar el tipo de datos. En cualquier caso, el tipo de dato puede ser establecido directamente para una column usando el tag <column> dentro de <table> (<view>, <query> o <select>).

Incluye conversores de datos. Por ejemplo, un valor numérico 0 ó 1 podría ser tratado como Boolean en Java; el status de la cuenta podría ser tratado como una "enum" de Java con cuatro valores, etc. Esto require de la codificación de una clase que implemente la interfaz TypeConverter.

HotRod puede usar tablas y vistas. Estas últimas puede ser actualizables (insert, delete, update) dependiendo de las reglas disponibles en cada base de datos. HotRod también puede usar secuencias y columnas identidades (by default o always) para generar valores de columnas.

Incluye tablas tipo enums. Éstas son tablas pequeñas (códigos, dimensiones, etc.) que no sufren modificaciones durante la operación normal de la aplicación, sino tan sólo entre releases. HotRod las carga enteramente como parte del código Java, con el fin de abaratar el costo de los queries.

HotRod incluye funcionalidad de "optimistic locking" que se activa en una tabla al agregar el tag <version-control-column>; si falla la actualización o eliminación de una fila en una tabla con esta característica lanza una RuntimeException, que podría iniciar el roll back de la transacción.

Los DAOs pueden exponer métodos para obtener números de secuencias, si dichas sequencias son de propósito general y no están relacionadas a una única tabla.

Para bases de datos masivas el archivo de configuración puede ser dividido en "fragmentos" administrados por equipos separados para evitar conflictos en el repositorio del código fuente.

El archivo de configuración puede incluir "facetas" que permiten realizar generaciones parciales. Esto puede ser útil para bases de datos masivas en donde se quiere generar sólo algunas tablas que han sido modificadas o agregadas.

La funcionalidad de HotRod está dividida en tres módulos:

- **HotRod CRUD**: Ofrece las operaciones básicas CRUD (y unas pocas más avanzadas) en base a una configuración mínima tal como se muestra en este ejemplo. Basta con indicar la lista de tablas y vistas, y HotRod genera todas las operaciones básicas automáticamente.

- **HotRod LiveSQL (experimental)**: permite ensamblar y ejecutar SELECTs simples y complejos por medio de métodos Java. Retorna lista de filas de propósito general, en donde cada fila es un Map de propiedades (String, Object).

- **HotRod Nitro**: Permite ejecutar queries complejos y/o de alto desempeño para explotar al máximo los recursos de la base de datos. Requiere incluir los comandos SQL en forma semi-automatizada en la configuracion de HotRod. Entro otros, esto permite agregar SQL dinámicos, parámetrizados, selects estructurados, asociaciones y colecciones. Este módulo está orientado a proveer alto desempeño durante la ejecución, y a simplificar la codificación estos. JOINs u otros SELECTs complejos son automáticamente tipificados y registrados.