

P4.3 CUENTAS CON

LISTA DE PEDIDOS

**MARIO JIMÉNEZ MARSET**

**ÍNDICE**

[1. ENUNCIADO - OBJETIVOS 3](#_Toc127544286)

[2. DESARROLLO – PROCEDIMIENTOS 3](#_Toc127544287)

# ENUNCIADO - OBJETIVOS

En esta práctica se pedía, a partir del ejemplo dado acerca de objetos foráneos, conectarse mediante a MySQL Workbench. Se pedía entonces comentar el código final y mostrar capturas de los resultados en el Workbench.

# DESARROLLO – PROCEDIMIENTOS

Se muestra el código comentado de las tres clases dadas.

Código Clase Account:

package ormlite3;

import com.j256.ormlite.dao.ForeignCollection;

import com.j256.ormlite.field.DatabaseField;

import com.j256.ormlite.field.ForeignCollectionField;

import com.j256.ormlite.table.DatabaseTable;

*/\*\**

*\* ejemplo de objeto Account el cual es persistente gracias al DAO*

*\*/*

@DatabaseTable(tableName = "accounts")

public class Account {

*//se crean las variables estáticas y constantes del nombre y la contraseña en los campos*

public static final String NAME\_FIELD\_NAME = "name";

public static final String PASSWORD\_FIELD\_NAME = "passwd";

*//se crea la variable id si el generateId es igual a verdadero*

@DatabaseField(generatedId = true)

private int id;

*//se crea el nombre de la columna si este es el de la variable anteriormente creada y no puede ser falso*

@DatabaseField(columnName = NAME\_FIELD\_NAME, canBeNull = false)

private String name;

*//se crea la contraseña si el nombre de la columna es igual al campo de la contraseña*

@DatabaseField(columnName = PASSWORD\_FIELD\_NAME)

private String password;

*//se crea la variable de tipo ForeignCollection, la cual retornará en su getter variables de tipo Order*

@ForeignCollectionField

private ForeignCollection<Order> orders;

*//todas las clases persistentes deben definir un constructor vacío*

Account() {}

*//se crean dos constructores diferentes con las variables anteriormente creadas*

public Account(String name) {

this.name = name;

}

public Account(String name, String password) {

this.name = name;

this.password = password;

}

public int getId() {

return id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

public ForeignCollection<Order> getOrders() {

return orders;

}

*//el método hashCode se ha sobrescrito para retornar la variable nombre con el método hashCode*

@Override

public int hashCode() {

return name.hashCode();

}

*//se sobrescribe el método equals con el objetivo de comparar la variable nombre con otro objeto casteado a Account*

@Override

public boolean equals(Object other) {

if (other == null || other.getClass() != getClass()) {

return false;

}

return name.equals(((Account) other).name);

}

}

Código Clase ForeignCollectionMain:

package ormlite3;

import java.util.List;

import com.j256.ormlite.dao.CloseableIterator;

import com.j256.ormlite.dao.Dao;

import com.j256.ormlite.dao.DaoManager;

import com.j256.ormlite.dao.ForeignCollection;

import com.j256.ormlite.jdbc.JdbcConnectionSource;

import com.j256.ormlite.support.ConnectionSource;

import com.j256.ormlite.table.TableUtils;

public class ForeignCollectionMain {

*//se crea la variable que almacenará el nombre de la base de datos, siendo esta de MySQL*

private final static String DATABASE\_URL = "dpruebaTres";

*//se crean las variables de tipo Dao, cuyo tipado es de las otras clases (Account y Order)*

private Dao<Account, Integer> accountDao;

private Dao<Order, Integer> orderDao;

public static void main(String[] args) throws Exception {

*//se llama al método doMain, pasándole como parámetro los argumentos*

new ForeignCollectionMain().doMain(args);

}

private void doMain(String[] args) throws Exception {

JdbcConnectionSource connectionSource = null;

try {

*//dentro de este método, se crea la conexión, a la cual se le pasa la url de conexión a MySQL junto con la variable anteriormente creada que almacenaba el nombre de la base de datos*

connectionSource = new JdbcConnectionSource("jdbc:mysql://localhost:3306/"+DATABASE\_URL,"root","mysql");

*//se llama al método que configura la base de datos y pasa esta conexión como parámetro*

setupDatabase(connectionSource);

*//se llama al método que lee y escribe datos*

readWriteData();

System.out.println("\n\nIt seems to have worked\n\n");

} finally {

*//se elimina la fuente de datos*

if (connectionSource != null) {

connectionSource.close();

}

}

}

private void setupDatabase(ConnectionSource connectionSource) throws Exception {

*//método donde se inicializan las variables llamando al método* createDao, pasando como parámetro la conexión creada y el .class predefinido

accountDao = DaoManager.createDao(connectionSource, Account.class);

orderDao = DaoManager.createDao(connectionSource, Order.class);

*//si se necesita crear la tabla (solo la primera ejecución del programa)*

TableUtils.createTable(connectionSource, Account.class);

TableUtils.createTable(connectionSource, Order.class);

}

private void readWriteData() throws Exception {

*//se crea una instancia de la clase Account pasando por parámetro el nombre creado*

String name = "Buzz Lightyear";

Account account = new Account(name);

*//con el método create, el objeto Account persiste en la base de datos*

accountDao.create(account);

*//se crea un objeto Order asociado con Account, pasando por parámetro las variables creadas*

int quantity1 = 2;

int itemNumber1 = 21312;

float price1 = 12.32F;

Order order1 = new Order(account, itemNumber1, price1, quantity1);

orderDao.create(order1);

*//se crea otro objeto Order*

int quantity2 = 1;

int itemNumber2 = 785;

float price2 = 7.98F;

Order order2 = new Order(account, itemNumber2, price2, quantity2);

orderDao.create(order2);

*//se crea una variable que recoge el id del objeto de la clase Account determinada*

Account accountResult = accountDao.queryForId(account.getId());

*//esta variable resultado, se le pasa el método para recoger los objetos Order y almacenarlos en una colección de tipo Order*

ForeignCollection<Order> orders = accountResult.getOrders();

*//se comprueban las operaciones*

CloseableIterator<Order> iterator = orders.closeableIterator();

try {

Order order = iterator.next();

order = iterator.next();

} finally {

iterator.close();

}

*//se crea un tercer objeto de tipo Order con variables diferentes*

int quantity3 = 50;

int itemNumber3 = 78315;

float price3 = 72.98F;

Order order3 = new Order(account, itemNumber3, price3, quantity3);

*//creado el objeto, se añade a la colección anterior*

orders.add(order3);

*//en una variable de tipo List, se consultan los objetos Order*

List<Order> orderList = orderDao.queryForAll();

}

}

Código Clase Order:

package ormlite3;

import com.j256.ormlite.field.DatabaseField;

import com.j256.ormlite.table.DatabaseTable;

*/\*\**

*\* ejemplo de objeto de la clase Order que es persistente gracias al DAO*

*\*/*

@DatabaseTable(tableName = "orders")

public class Order {

*//se crea una variable estática y constante el cual es el id de Account*

public static final String ACCOUNT\_ID\_FIELD\_NAME = "account\_id";

*//se crea una variable id solo si generateId es verdadero*

@DatabaseField(generatedId = true)

private int id;

*//se crea una instancia de Account si foreign es verdadero y la variable nombre de columna es igual a la variable anteriormente creada*

@DatabaseField(foreign = true, columnName = ACCOUNT\_ID\_FIELD\_NAME)

private Account account;

*//se crean tres variables enteras y float (las cuales se pasaran a la hora de crear objetos Order en ForeignMain)*

@DatabaseField

private int itemNumber;

@DatabaseField

private int quantity;

@DatabaseField

private float price;

*//todas las clases persistentes deben definir un constructor vacío*

Order() {}

*//se crea un constructor con las variables anteriormente creadas*

public Order(Account account, int itemNumber, float price, int quantity) {

this.account = account;

this.itemNumber = itemNumber;

this.price = price;

this.quantity = quantity;

}

*//se crean los getters y setters de las variables*

public int getId() {

return id;

}

public Account getAccount() {

return account;

}

public void setAccount(Account account) {

this.account = account;

}

public int getItemNumber() {

return itemNumber;

}

public void setItemNumber(int itemNumber) {

this.itemNumber = itemNumber;

}

public int getQuantity() {

return quantity;

}

public void setQuantity(int quantity) {

this.quantity = quantity;

}

public float getPrice() {

return price;

}

public void setPrice(float price) {

this.price = price;

}

}