**GROUP 5**

**Hibernate DB**

Nelson Liang

Victor Li

Udit Manocha

Steven Gonzalez

**Overview**

The purpose of this project is to implement an Object-Relational Database (ORDB) using the Hibernate Java Object-Relational Mapping tool. This project will demonstrate Insertion, aggregation and queries on the ORDB. A Java Object of type salesTansaction will represent a sales-transaction, each salesTransaction Object will be stored into the ORDB in the Sales table. Using the hibernate Mapping tool, we will also discuss how to retrieve single salesTransaction, retrieve a salesTransaction over a given time interval and perform aggregate operations on the salesTransaction document.

**Technologies**

For the purpose of this project, the following technologies were used to implement our Object-Relational Database:

MySQL – SQL based Relational Database Management System (RDBMS).

Hibernate – Java Object-Relational Mapping tool – allows us to store objects in our RDBMS.

Eclipse – Development environment that is used to implement, execute and test our ORDB.

**Implementation**

The file hibernate.cfg.xml defines the configuration properties of our Object-Relational Database. This file tells our ORDB which database dialect and connection to use. For this project, we used MySQL for the hibernate-dialect and connection driver. The credentials for our databases username and password were also defined and the option to create the salesTransaction table was also implemented.

The salesTransaction class which defines a java object of type salesTransaction. These salesTransaction objects are made up of: a date, productName, Quantity, UnitCost and totalCost. This class also defines accessor and mutate functions which are used in during the creation and retrieval of information from salesTransaction objects.

The main class, configures the hibernate.cfg.xml file for the project and a service registry is instantiated in order to open a session. Once the session was configured, an arraylist was used to make a list that would be added to the salesTransaction table, 6 transaction were added to the list and passed to the addSalesTransaction methods. In this method, each salesTransaction is added to the table and the changes are committed. The main class also performs the queries and handles data retrieval for testing the ORDB.

**Queries**

/\* find sales transactions between two time intervals \*/

FROM Sales WHERE Date BETWEEN :date1 AND :date2;

/\* print a single transaction based on the date \*/

FROM Sales WHERE Date = :Date;

**Aggregation**

/\* aggregate operation on sales transaction object find MIN(TotalCost) \*/

SELECT MIN(sales.TotalCost) FROM Sales sales;

/\* aggregate operation on sales transaction object find MAX(TotalCost) \*/

SELECT MAX(sales.TotalCost) FROM Sales sales;

/\* aggregate operation on sales transaction object find SUM(TotalCost) \*/

SELECT SUM(sales.TotalCost) FROM Sales sales;

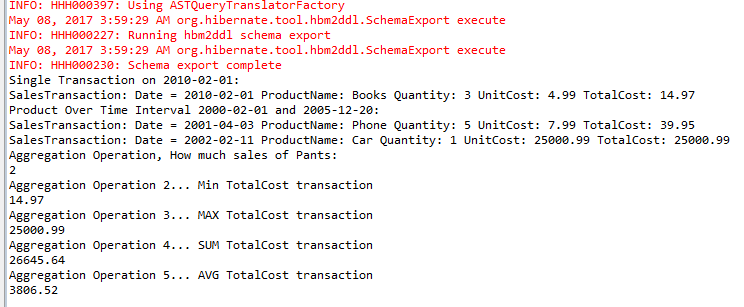
/\* aggregate operation on sales transaction object find AVG(TotalCost) \*/

SELECT AVG(sales.TotalCost) FROM Sales sales;

/\* aggregate operation on sales transaction How much sales of a given product item \*/

SELECT count(sales.ProductName) FROM Sales sales WHERE sales.ProductName = :ProductName;

**Output**



**Code – Main.java class**

import java.io.File;

import java.text.SimpleDateFormat;

import java.util.ArrayList;

import java.util.Date;

import java.util.List;

import org.hibernate.HibernateException;

import org.hibernate.Query;

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.Transaction;

import org.hibernate.cfg.Configuration;

import org.hibernate.service.ServiceRegistry;

import org.hibernate.service.ServiceRegistryBuilder;

public class Main {

@SuppressWarnings("deprecation")

public static void main(String[] args) {

Configuration configuration = new Configuration().configure()

.addAnnotatedClass(SalesTransactions.class);

ServiceRegistry serviceRegistry = new ServiceRegistryBuilder()

.applySettings(configuration.getProperties())

.buildServiceRegistry();

SessionFactory sessionFactory = configuration

.buildSessionFactory(serviceRegistry);

// store list of Sale transaction to add to sales table

// java date constructor: Date(year + 1900, month + 1, day)

// i.e Date(110, 4, 5) => 2010-05-05

ArrayList<SalesTransactions> list = new ArrayList<SalesTransactions>();

list.add(new SalesTransactions(new Date(110, 1, 1), "Books", 3, 4.99));

list.add(new SalesTransactions(new Date(101, 3, 3), "Phone", 5, 7.99));

list.add(new SalesTransactions(new Date(102, 1, 11), "Car", 1, 25000.99));

list.add(new SalesTransactions(new Date(117, 1, 5), "Shoes", 10, 99.99));

list.add(new SalesTransactions(new Date(117, 7, 7), "Shirt", 3, 9.99));

list.add(new SalesTransactions(new Date(117, 3, 20), "Pants", 7, 39.99));

list.add(new SalesTransactions(new Date(117, 2, 20), "Pants", 7, 39.99));

addSalesTransactions(sessionFactory, list);

// find sales transaction that happened on Date(110, 1, 1)

Date date = new Date(110, 1, 1);

SingleTransaction(sessionFactory, date);

// Retrieve sales transactions for a given product over a given time

// interval

// in this case (100, 1, 1) and (105, 11, 20)

Date date1 = new Date(100, 1, 1);

Date date2 = new Date(105, 11, 20);

ProductOverTimeInterval(sessionFactory, date1, date2);

// aggregate operations on sales transaction objects

AggregateOperation(sessionFactory, "Pants");

AggregateTwo(sessionFactory);

AggregateThree(sessionFactory);

AggregateFour(sessionFactory);

AggregateFive(sessionFactory);

}

/\*\*

\* aggregate operation on sales transaction objects How much sales of

\* a given product item

\*

\* @param sessionFactory: creating hibernate sessions to query

\* @param productName: name of product

\*/

public static void AggregateOperation(SessionFactory sessionFactory,

String productName) {

Session session = sessionFactory.openSession();

String hql = "SELECT count(sales.ProductName) FROM Sales sales WHERE sales.ProductName = :ProductName";

Query query = session.createQuery(hql);

query.setParameter("ProductName", productName);

List results = query.list();

System.out.println("Aggregation Operation, How much sales of "

+ productName + ":");

Number number = (Number) results.get(0);

System.out.println(number.intValue());

session.close();

}

/\*\*

\* aggregate operation on sales transaction object find Min(TotalCost)

\* salesTransaction

\*

\* @param sessionFactory: creating hibernate sessions to query

\*/

public static void AggregateTwo(SessionFactory sessionFactory) {

Session session = sessionFactory.openSession();

String hql = "SELECT MIN(sales.TotalCost) FROM Sales sales ";

Query query = session.createQuery(hql);

List results = query.list();

System.out.println("Aggregation Operation 2... Min TotalCost transaction ");

double number = (double) results.get(0);

System.out.format("%.2f", number);

System.out.println();

session.close();

}

/\*\*

\* aggregate operation on sales transaction object find MAX(TotalCost)

\* salesTransaction

\*

\* @param sessionFactory: creating hibernate sessions to query

\*/

public static void AggregateThree(SessionFactory sessionFactory) {

Session session = sessionFactory.openSession();

String hql = "SELECT MAX(sales.TotalCost) FROM Sales sales ";

Query query = session.createQuery(hql);

List results = query.list();

System.out.println("Aggregation Operation 3... MAX TotalCost transaction ");

double number = (double) results.get(0);

System.out.format("%.2f", number);

System.out.println();

session.close();

}

/\*\*

\* aggregate operation on sales transaction object find MAX(TotalCost)

\* salesTransaction

\*

\* @param sessionFactory: creating hibernate sessions to query

\*/

public static void AggregateFour(SessionFactory sessionFactory) {

Session session = sessionFactory.openSession();

String hql = "SELECT SUM(sales.TotalCost) FROM Sales sales ";

Query query = session.createQuery(hql);

List results = query.list();

System.out.println("Aggregation Operation 4... SUM TotalCost transaction ");

double number = (double) results.get(0);

System.out.format("%.2f", number);

System.out.println();

session.close();

}

/\*\*

\* aggregate operation on sales transaction object find MAX(TotalCost)

\* salesTransaction

\*

\* @param sessionFactory: creating hibernate sessions to query

\*/

public static void AggregateFive(SessionFactory sessionFactory) {

Session session = sessionFactory.openSession();

String hql = "SELECT AVG(sales.TotalCost) FROM Sales sales ";

Query query = session.createQuery(hql);

List results = query.list();

System.out.println("Aggregation Operation 5... AVG TotalCost transaction ");

double number = (double) results.get(0);

System.out.format("%.2f", number);

System.out.println();

session.close();

}

/\*\*

\* find sales transactions between two time intervals

\*

\* @param sessionFactory: creating hibernate sessions to query

\* @param date1: sales transactions after date1

\* @param date2: sales transactions before date2

\*/

public static void ProductOverTimeInterval(SessionFactory sessionFactory,

Date date1, Date date2) {

Session session = sessionFactory.openSession();

String hql = "FROM Sales WHERE Date BETWEEN :date1 AND :date2";

Query query = session.createQuery(hql);

query.setParameter("date1", date1);

query.setParameter("date2", date2);

List<SalesTransactions> result = query.list();

System.out.println("Product Over Time Interval "

+ new SimpleDateFormat("yyyy-MM-dd").format(date1) + " and "

+ new SimpleDateFormat("yyyy-MM-dd").format(date2) + ":");

print(session, query.list());

}

/\*\*

\* print a single transaction based on the date

\*

\* @param sessionFactory: creating hibernate sessions to query

\* @param date: date in which transaction is in

\*/

public static void SingleTransaction(SessionFactory sessionFactory,

Date date) {

Session session = sessionFactory.openSession();

String hql = "FROM Sales WHERE Date = :Date";

Query query = session.createQuery(hql);

query.setParameter("Date", date);

System.out.println("Single Transaction on "

+ new SimpleDateFormat("yyyy-MM-dd").format(date) + ":");

print(session, query.list());

}

/\*\*

\* prints the result list on the queries

\*

\* @param session: given session to close

\* @param result: result list on the query

\*/

public static void print(Session session, List<SalesTransactions> result) {

SalesTransactions salesTransaction = null;

for (SalesTransactions s : result) {

System.out.println("SalesTransaction: " + "Date = " + s.getDate()

+ " ProductName: " + s.getProductName() + " Quantity: "

+ s.getQuantity() + " UnitCost: " + s.getUnitCost()

+ " TotalCost: " + s.getTotalCost());

}

session.close();

}

/\*\*

\* adds sales transactions to sales table

\*

\* @param sessionFactory: creating hibernate sessions

\* @param list: list of sales transactions to add to sales table

\*/

public static void addSalesTransactions(SessionFactory sessionFactory,

ArrayList<SalesTransactions> list) {

Session session = sessionFactory.openSession();

Transaction tx = null;

try {

tx = session.beginTransaction();

for (SalesTransactions s : list) {

session.save(s);

}

tx.commit();

} catch (HibernateException e) {

if (tx != null)

tx.rollback();

e.printStackTrace();

} finally {

session.close();

}

}

}

**Code – SalesTransaction.java (Object)**

import java.util.Date;

import javax.persistence.\*;

@Entity(name = "Sales")

public class SalesTransactions {

@Id

@Column(name = "Date")

@Temporal(TemporalType.DATE)

private Date Date;

@Column(name = "ProductName")

private String ProductName;

@Column(name = "Quantity")

private int Quantity;

@Column(name = "UnitCost")

private double UnitCost;

@Column(name = "TotalCost")

private double TotalCost;

public Date getDate() {

return Date;

}

public void setDate(Date Date) {

this.Date = Date;

}

public String getProductName() {

return ProductName;

}

public void setProductName(String productName) {

ProductName = productName;

}

public int getQuantity() {

return Quantity;

}

public void setQuantity(int quantity) {

Quantity = quantity;

}

public double getUnitCost() {

return UnitCost;

}

public void setUnitCost(double unitCost) {

UnitCost = unitCost;

}

public double getTotalCost() {

return TotalCost;

}

public void setTotalCost(double totalCost) {

TotalCost = totalCost;

}

public SalesTransactions() {

}

public SalesTransactions(Date Date, String ProductName, int Quantity,

double UnitCost) {

this.Date = Date;

this.ProductName = ProductName;

this.Quantity = Quantity;

this.UnitCost = UnitCost;

this.TotalCost = Quantity \* UnitCost;

}

}

**Code – hibernate.cfg.xml (hibernate configuration/mapping)**

<?xml version=*"1.0"* encoding=*"utf-8"*?>

<!DOCTYPE hibernate-configuration SYSTEM

"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<property name=*"hibernate.dialect"*>

org.hibernate.dialect.MySQLDialect

</property>

<property name=*"hibernate.connection.driver\_class"*>

com.mysql.jdbc.Driver

</property>

<!-- Assume cs157bproject is the database name -->

<property name=*"hibernate.connection.url"*>

jdbc:mysql://localhost/cs157bproject

</property>

<!-- Change MySQL credentials -->

<property name=*"hibernate.connection.username"*>

root

</property>

<property name=*"hibernate.connection.password"*>

cs157a

</property>

<!-- automatically creates the sales table mapped in SalesTransactions.java -->

<property name=*"hbm2ddl.auto"*>create</property>

</session-factory>

</hibernate-configuration>