Database Management Systems

Project Report

**Martin M. Taheri**

(mtaheri1@gmu.edu)

*Table of Contents*

1. Concept ............................................................................ 4

2. Design ............................................................................ 4

3. Implementation ............................................................................ 5

4. Result and Challenges ............................................................................ 6

Appendix:

A. ER-Models ............................ 7

B. SQL Schema and Insertions ............................ 9

C. Code samples ............................ 12

D. Output samples ............................ 14

*1. Concept*

*1.1 The Goal:* To design and implement a database for a deal aggregator site similar to LivingSocial1 . At the back-end, the database (Oracle 10g\SQL) manages all of the data, and provide a command-line interface at the front-end for the users using Java and JDBC.

*1.2 Functionality:*This includes variety of queries and browsing capabilities for the user. For example, the user is able to browse current deals, search by date, deal type, merchant, location, purchase history, and customer. Also the user shall be able to update the database by inserting and deleting tuples.

*2. Design*

*2.1 The Approach:* In order to facilitate the requirements as outlined above, I decided to create a relation for each major entity and have constraints or dependencies when joining different relations. For example, I decided to create a Customer relation, containing only customer related attributes such as Name, Email, Address, etc. I choose Email as the unique identifier (Primary Key) for this relation, meaning no two customers could have the same email address. This ensures that the tuples in my table are uniquely identifiable which alleviates redundancy issues, conforms to uniqueness and completeness of entries, as well as facilitating simple and fast searching. Some entities need to be able to access Customer table in order to store additional related information for example each customer could have their own particular deals that they favor over the others. Therefore, I would store these preferences in a separate table, namely, Interest. Also, I insert a foreign key constrained for the interest table to reference the email field from the customer table, this ensures the two tables would be joinable without containing any redundant information. Doings so will also ensure that the tables are in Boyce-Codd normal form (BCNF)2, eliminated any anomalies that may arise from redundancy on functional dependencies.

*2.2 Entities and Relationships:* Following the mentioned approach I divided up the task by created a table for each entity and ensuring the relationships I desired by adding dependency constrains on each table. For a complete list and diagram of all the entities, their attributes and relationship view the entity relationship model3.

1. www.livingsocial.com.

2.Codd, E. F. "Recent Investigations into Relational Data Base Systems." IBM Research Report RJ1385 (April 23, 1974). Republished in *Proc. 1974 Congress* (Stockholm, Sweden, 1974). New York, N.Y.: North-Holland (1974).

3. Appendix A, *page 7*

*3. Implementation*

*3.1 Database Schema:* There are a total of eleven tables for this project. All of which are created using, *CREATE TABLE ()* SQL command. I use standard SQL types to create the corresponding attributes for each table. For instance,  *INTEGER* and *VARCAH2* types are used to represent customer age and name, respectively. For a complete list of tables, their attributes, and corresponding data representation refer to the SQL Schema4. This information along with test data to insert into our database is located in *input.txt* file and is parsed at the start of the program execution in order to automatically create and populate tables for testing purposes.

*3.2 Object-oriented approach:* In order to provide the user with a medium to interface with my database, the program had to be written in a high level language. I chose Java to make use of the object oriented functionalities. The main libraries in use are the standard IO and Utilities libraries from JDK v.1.7. It was clear to me from the start that I had to create and independent object for each of my tables. Each attribute for a table is a private variable with getter and setter methods residing in the corresponding class. Each have public methods to insert tuples, delete tuples, check if tuple exist, find a tuple based on a given criteria, and to display the table. Refer to the included code snippet5 for more details. Furthermore, I created a wrapper class named *proj.java*, that contains the main method of the program as well as object initialization for all other classes. The main method acts as the front-end interface where the user can choose to modify the database, or query for deals through a series of menu options that will be displayed at the command line. Refer to the appendix for a sample output6. A *makefile* is used along with *JCC* as the compiler and *GDB* as the debugger to link and compile all eleven classes to the wrapper class, finally creating the executable file.

*3.3 DBMS Connection:* I use the ORACLE JDBC driver to interact with the database management system directly from within my program. This allows me to open a direct connection with the server that hosts the DBMS using my DBA username and password. JAVA JDBC also includes useful libraries that I use throughout the program. For example, I use java.sql.DatabaseMetaData to retrieve meta data information on the tables stored, in order to find a particular table to operate on. Java.sql.Statement and java.sql.ResultSet are used whenever I want to execute my hardcoded SQL query string on the table, retrieving the results and converting them to a Java array type, allowing me to use them as standard java types in the program or to be displayed as final query results to the user.

*4. Result and Challenges*

*4.1 Final Outcome:* This software allows users to create tables, view tables, insert tuples, remove tuples, and search for entries inside of an ORACLE database management system through a command line menu-based interface. Furthermore, users are able to narrow down their search for deals based on merchant name, location, open deals, etc. The final program includes more than seven thousand lines of code, and took over two weeks to complete.

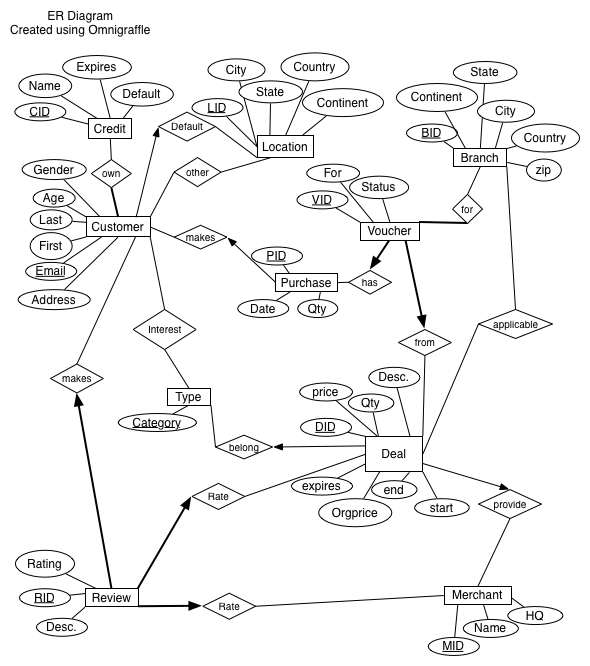
*4.2 Challenges:* Interfacing with ORACLE DBMS through a high-level program like JAVA is a new experience for me. Since SQL and JAVA differ greatly grammatically as languages, it was a challenging task, initially, to compose SQL strings inside of JAVA and execute them successfully through the JDBC driver. After a few tries, however, I was able to conquer this challenge and spent the rest of my time on other high priority items. I am glad that I learned a great deal through this project, having met and exceeded all the objectives on time.

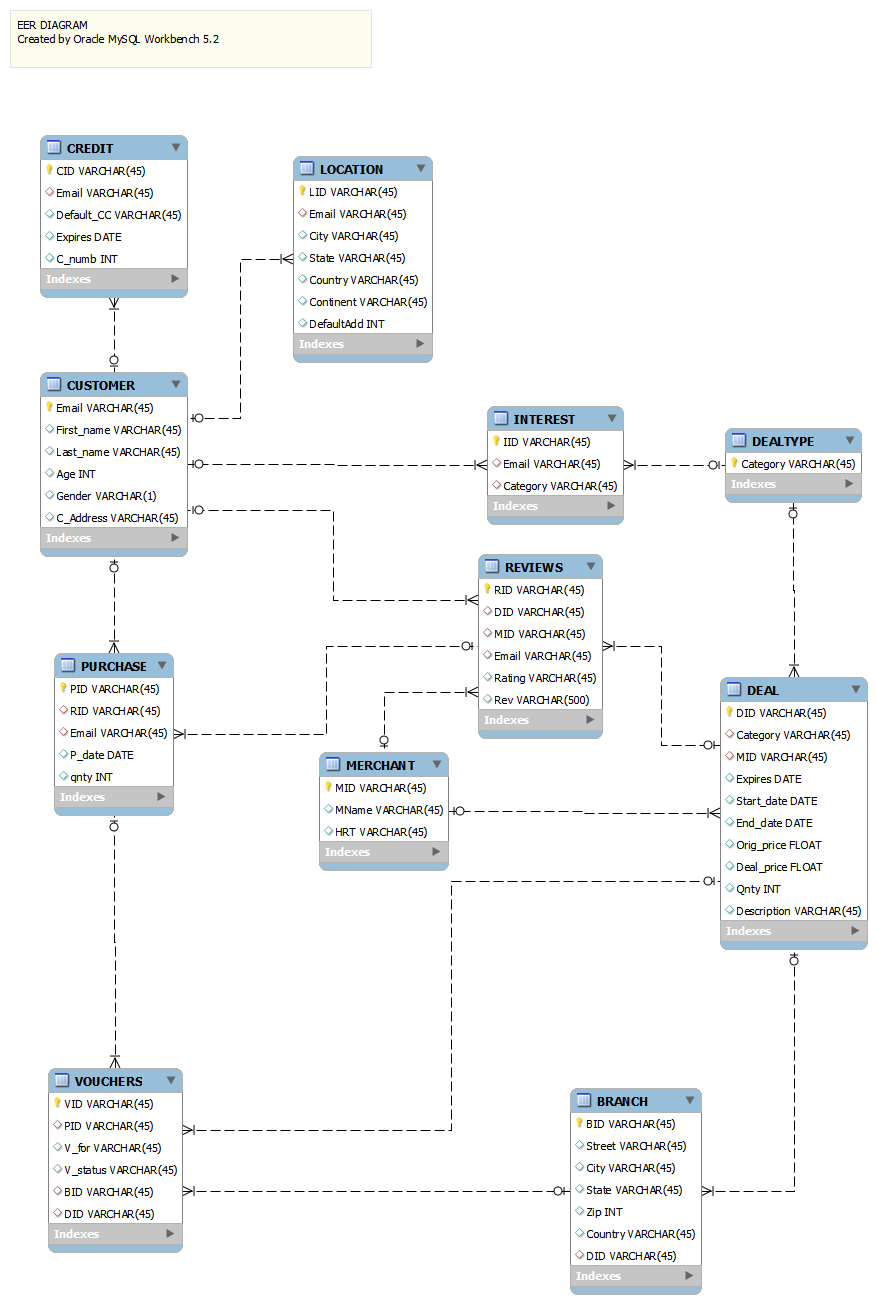
4. Appendix B, *page 9*

5. Appendix C, *page 12*

6. Appendix D, *page 14*

*Appendix A*





*Appendix B*

--Database Schema

--Customer table

CREATE TABLE CUSTOMER (

email VARCHAR2(25) ,

fname VARCHAR2(25) ,

lname VARCHAR2(25) ,

Age INTEGER ,

GENDER CHAR(1) NOT NULL CHECK(GENDER IN('M','F') ,

address VARCHAR2(25) ,

PRIMARY KEY (email) );

--Credit table

CREATE TABLE CREDIT (

CID VARCHAR2(25) ,

Email VARCHAR2(25),

Defaultcc VARCHAR2(25) ,

Expires DATE ,

cc\_number INTEGER ,

PRIMARY KEY (CID) ,

FOREIGN KEY (email)

REFERENCES CUSTOMER(email));

--Dealtype table

CREATE TABLE DEALTYPE (

Category VARCHAR2(25),

PRIMARY KEY (Category));

--Merchant table

CREATE TABLE MERCHANT (

mid VARCHAR2(25) ,

name VARCHAR2(25) ,

hq VARCHAR2(25) ,

PRIMARY KEY (mid));

--Deal table

CREATE TABLE DEAL (

did VARCHAR2(25) ,

Category VARCHAR2(25) ,

mid VARCHAR2(25) ,

expires DATE ,

start\_date DATE ,

end\_date DATE ,

org\_price FLOAT ,

deal\_price FLOAT ,

Qty INTEGER ,

Description VARCHAR2(25) ,

PRIMARY KEY (did) ,

FOREIGN KEY (Category )

REFERENCES DEALTYPE (Category ),

FOREIGN KEY (mid )

REFERENCES MERCHANT (mid));

--Review table

CREATE TABLE REVIEW (

rid VARCHAR2(25) ,

did VARCHAR2(25) ,

mid VARCHAR2(25) ,

email VARCHAR2(25) ,

Rating INTEGER ,

description VARCHAR2(500) ,

PRIMARY KEY (rid) ,

FOREIGN KEY (did)

REFERENCES DEAL (did),

FOREIGN KEY (mid)

REFERENCES MERCHANT (mid),

FOREIGN KEY (email)

REFERENCES CUSTOMER (email));

--Purchase table

CREATE TABLE PURCHASE (

PID VARCHAR2(25) ,

RID VARCHAR2(25) ,

qty INTEGER ,

EMAIL VARCHAR2(25) ,

p\_date DATE ,

PRIMARY KEY (PID) ,

FOREIGN KEY (RID)

REFERENCES REVIEWS (RID),

FOREIGN KEY (EMAIL)

REFERENCES CUSTOMER (EMAIL));

--Branch table

CREATE TABLE BRANCH (

bid VARCHAR(25) ,

street VARCHAR(25) ,

City VARCHAR(25) ,

State VARCHAR(25) ,

Country VARCHAR(25) ,

DID VARCHAR(25) ,

Zip INTEGER ,

PRIMARY KEY (bid) ,

FOREIGN KEY (DID)

REFERENCES DEAL (DID));

--Voucher table

CREATE TABLE VOUCHERS (

vid VARCHAR(25) ,

pid VARCHAR(25) ,

bid VARCHAR(25) ,

did VARCHAR(25) ,

vfor VARCHAR(25) ,

status VARCHAR(25) ,

PRIMARY KEY (vid) ,

FOREIGN KEY (pid)

REFERENCES PURCHASE (pid),

FOREIGN KEY (bid)

REFERENCES BRANCH (bid),

FOREIGN KEY (did)

REFERENCES DEAL (did));

--Location Table

CREATE TABLE LOCATION (

lid VARCHAR(25) ,

email VARCHAR(25) ,

City VARCHAR(25) ,

State VARCHAR(25) ,

Country VARCHAR(25) ,

Continent VARCHAR(25) ,

Defaultadd INTEGER ,

PRIMARY KEY (lid) ,

FOREIGN KEY (email)

REFERENCES CUSTOMER (email));

--Interest Table

CREATE TABLE INTEREST (

IID VARCHAR2(25) ,

Category VARCHAR2(25) ,

EMAIL VARCHAR2(25) ,

PRIMARY KEY (IID) ,

FOREIGN KEY (EMAIL)

REFERENCES CUSTOMER (EMAIL),

FOREIGN KEY (Category)

REFERENCES DEALTYPE (Category));

--for insertions refer to input.txt

*Appendix C*

/\*\*

\* **@throws** SQLException

\*/

**public** **void** insertData () **throws** SQLException

{

// get the connection

Connection connection = getConnection();

// create new table

**if** (!**this**.doesTableExist(connection))

{

// create the table

System.*out*.println("CUSTOMERS Table doesn't exist. Creating new instance.....");

createTable(connection);

}

// create the INSERT SQL

StringBuffer sbInsert = **new** StringBuffer();

sbInsert.append(" INSERT INTO CUSTOMER (email, fname, lname, AGE, GENDER, address) ");

sbInsert.append(" VALUES ");

sbInsert.append(" ('" + email + "', '" + fname + "','" + lname + "','" + age + "','" + gender + "', '" + address + "')");

// create the statement

Statement statement = connection.createStatement();

**try**

{

// Insert the data

statement.executeUpdate (sbInsert.toString());

} **catch** (SQLException e)

{

**throw** e;

} **finally**

{

statement.close();

close(connection);

}

}

/\*\*

\* **@param** mail

\* **@throws** SQLException

\*/

**public** **static** **boolean** doesmailexist(String mail) **throws** SQLException

{

Customer m=**new** Customer();

Connection con=m.getConnection();

**boolean** a=**false**;

//creat SQL statement

StringBuffer sbselect = **new** StringBuffer();

sbselect.append(" SELECT EMAIL ");

sbselect.append(" FROM CUSTOMER");

sbselect.append(" WHERE EMAIL=");

sbselect.append("'" + mail + "'");

Statement statement=**null**;

ResultSet rs=**null**;

ArrayList collection = **new** ArrayList();

**try**{

statement=con.createStatement();

rs=statement.executeQuery(sbselect.toString());

**if** ( rs.next()){

a=**true**;

}

}**catch**(SQLException e)

{

**throw** e;

}**finally**

{

rs.close();

statement.close();

m.close(con);

}

**return** a;

}//end of checkmail

*Appendix D*

\_\_\_\_\_\_MENU\_\_\_\_\_\_

1.Display Tables

2.Modify Tables

3.Search for Deals

4.Open Query

5.Exit

Enter: 2

----Select DB----

1. Customer

2. Deal

3. Merchant

4. Purchase

Enter#: 1

----Modify Menu----

1.Add record

2.Delete record

Enter#: 1

Enter Customer Email: user1@domain.com

Enter First Name: John

Enter Last Name: Smith

Enter Age: 27

Enter Gender(M or F): M

Enter Address: SomewhereSt.

Inserting Customer...

Add agian?(y/n): n

done

\_\_\_\_\_\_MENU\_\_\_\_\_\_

1.Display Tables

2.Modify Tables

3.Search for Deals

4.Open Query

5.Exit

Enter: 1

----Select DB----

1. Customer

2. Deal

3. Merchant

4. Purchase

5. Branch

6. Credit

7. Voucher

8. Dealtype

9. Interest

10.Location

11.Review

Enter#: 1

email fname lname Age Gender address

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

11

A@domain.com John Smith 18 M Somewhere

B@domain.com Jane Smith 18 F Somewhere

C@domain.com Joe Smith 18 M Somewhere

D@domain.com Jack Smith 18 M Somewhere

E@domain.com Jill Smith 18 F Somewhere

F@domain.com Jon Smith 18 M Somewhere

G@domain.com Jonathan Smith 18 M Somewhere

H@domain.com Jessica Smith 25 F Somewhere

I@domain.com Jasmin Smith 25 F Somewhere

J@domain.com George Smith 50 M Somewhere

user1@domain.com John Smith 27 M SomewhereSt.