EECS 484 W13 Project 2
Querying Fakebook Database with Java and JDBC

Due on October 9, 2014 by 11:55PM

Overview

Project 1 focused primarily on database design. In Project 2, you will focus on writing SQL queries. In addition, you will embed your SQL queries into Java (using JDBC) to implement a "Fakebook Oracle," a standalone program that answers several queries about the Fakebook database. For this project, you will use our "official" schema, rather than the schema you designed in Project 1. You will have access to our public fake data.

1. Tables

For this project, your schema will consist of the following twelve tables:

1. <prefix>.<DataType>_USERS
2. <prefix>.<DataType>_FRIENDS
3. <prefix>.<DataType>_CITIES
4. <prefix>.<DataType>_PROGRAMS
5. <prefix>.<DataType>_USER_CURRENT_CITY
6. <prefix>.<DataType>_USER_HOMETOWN_CITY
7. <prefix>.<DataType>_EDUCATION
8. <prefix>.<DataType>_USER_EVENTS
9. <prefix>.<DataType>_PHOTOS
10. <prefix>.<DataType>_ALBUMS
11. <prefix>.<DataType>_TAGS
12. <prefix>.<DataType>_PARTICIPANTS

Public Fake Data:

<DataType> should be replaced with "PUBLIC" to access the fake data tables.

The public fake data tables are stored in the GSI's account (yjtang). Therefore, you should use the GSI's account name (yjtang) as <prefix> to access the public tables directly within sqlplus for testing your queries. For example, to access the public USERS table, you should refer to the table name as YJTANG.PUBLIC_USERS. For the Java files provided to you, the above table names are already pre-configured in the given code.

2. Files Provided to You

You are provided 3 java files: 'TestFakebookOracle.java', 'FakebookOracle.java' and 'MyFakebookOracle.java'. You only need to turn in MyFakebookOracle.java in the end.

1. TestFakebookOracle.java

This file provides the main function for running the program. You should only modify the following 3 static variables, replacing them with your own information.

```
public class TestFakebookOracle {

    static String dataType = "PUBLIC";
    static String oracleUserName = "username"; //replace with your Oracle account name
    static String password = "password"; //replace with your Oracle password


    public static void main(String[] args) {
```

2. FakebookOracle.java

DO NOT modify this file. You can have a look at the content if you are curious.

This class defines the query functions (which will be introduced later) as abstract functions, which you must implement for Project 2. It also defines some useful data structures and provides formatted print functions.

3. MyFakebookOracle.java

This is a subclass of FakebookOracle, in which you must implement the query functions. You should ONLY fill in the body for each of those functions, and DO NOT modify anywhere else.

```
public class MyFakebookOracle extends FakebookOracle {

    static String prefix = "yjtang.";
```

In this project, you only need to store the results of the queries in our predefined data structures (which we have provided as member variables in the class). You don't need to worry about any output formatting. In the base class FakebookOracle.java, a set of print functions have been provided for you to view the query results.

The class contains parameterized names for the tables you need to use in your queries, and they are constructed in the class constructor as show in the following lines of code. You should always use the corresponding variable when you are referring to a table in the SQL statement to be executed through JDBC. For example, you should use the variable cityTableName instead of using a constant value such as 'public_cities' in your java code.

For database connection, you should use the object oracleConnection.

```
// DO NOT modify this constructor
public MyFakebookOracle(String u, Connection c) {
    super();
    String dataType = u;
    oracleConnection = c;
    // You will use the following tables in your Java code
    cityTableName = prefix+dataType+"_CITIES";
    userTableName = prefix+dataType+"_USERS";
    friendsTableName = prefix+dataType+"_FRIENDS";
    currentCityTableName = prefix+dataType+"_USER_CURRENT_CITY";
    hometownCityTableName = prefix+dataType+"_USER_HOMETOWN_CITY";
    programTableName = prefix+dataType+"_PROGRAMS";
    educationTableName = prefix+dataType+"_EDUCATION";
    eventTableName = prefix+dataType+"_USER_EVENTS";
    albumTableName = prefix+dataType+"_ALBUMS";
    photoTableName = prefix+dataType+"_PHOTOS";
    tagTableName = prefix+dataType+"_TAGS";
}
```

3. solution-public.txt

This file contains the reference result from queries that are given in Section 3 when you will run your Java program in Part 3. That can help you validate whether your queries generate the correct results on the same database. Note that we do the testing on a hidden database. So, make sure that your queries are designed to work in general, not just on sample data. Think carefully about the semantics of your queries since it may not be always possible to test them in all scenarios and you often will not have the benefit of knowing the correct answers in practice. Here you have one set of answers on a sample dataset.

3. Queries (90 points)

There are 10 total queries (Query0 to Query9). Query0 is provided to you as an example, and you need to implement the other 9. The points are evenly distributed (10 points per query).

However, the queries vary tremendously in terms of difficulty. If you get stuck on a harder query, try an easier one first, and then come back to the tough one. For all these queries, sample answers on the given data are available in the attached zip file. If the English description is ambiguous, please look at the sample answers.

Also, for all these queries, you should try to do all the work within SQL to the extent possible. For example, if a query requires you to present the data in sorted order, use ORDERED BY in your query rather than retrieving the result and then sorting it within Java.

Also, the grading program we use does limit the time it waits for a query. If a query appears to be taking too much time for you (order of several minutes) you should think about rewriting it in a different way to make it faster. Usually, nested queries are more expensive to run.

Query0 (0 points): Find information about month of birth

This function has been implemented for you in MyFakebookOracle.java, so that you can use it as an example. The function computes the month in which the most friends were born and the month in which the fewest friends were born. The names of these friends are also retrieved.

The sample function uses the Connection object oracleConnection to build a Statement object. Using the Statement object, it issues a SQL query, and retrieves a ResultSet. It iterates over the result set, and stores the necessary results in a Java object. Finally, it closes both the Statement and the ResultSet objects.

Query1 (10 points): Find information about names

The next query asks you to find information about user names, including (1) The longest last name, (2) The shortest last name, and (3) The most common last name. If there are ties, you should include all the matches in the result.

The following code snippet illustrates the data structures that should be constructed. However, it is up to you to add your own JDBC query to answer the question correctly.

```java
@Override
// ***** Query 1 *****
// Find information about friend names:
// (1) The longest last name (if there is a tie, include all in result)
// (2) The shortest last name (if there is a tie, include all in result)
// (3) The most common last name, and the number of times it appears (if there is a
//     tie, include all in result)
//
public void findNameInfo() throws SQLException { // Query1
    // Find the following information from your database and store the information as shown
    this.longestLastNames.add("JohnJacobJingleheimerSchmidt");
    this.shortestLastNames.add("Ng");
    this.shortestLastNames.add("Fu");
    this.shortestLastNames.add("Wu");
    this.mostCommonLastNames.add("Wang");
    this.mostCommonLastNames.add("Smith");
    this.mostCommonLastNamesCount = 10;
}
```

Query2 (10 points): Find “lonely” friends

The next query asks you to find information about all users who have no friends in the network. Again, you will place your results into the provided data structures. The sample code in MyFakebookOracle.java illustrates how to do this.

Query3 (10 points): Find “homebodies”

The next query asks you to find information about all friends who still live in their hometowns. In other words, the current_city associated with these users should be equal to the hometown_city. (Neither should be null.) You will place your result into the provided data structures.

Query4 (10 points): Find information about photo tags

For this query, you should find the top n photos that have the most tagged users. You will also need to retrieve information about each of the tagged users. If there are ties (i.e., photos with the same number of tagged users), then choose the photo with smaller id first (this will be string lexicographic ordering since the datatypes are varchars. For instance, 10 will be less than 2 in such an ordering).

Query5 (10 points): Find friends to set up on dates

For this task, you should find the top n “match pairs” according to the following criteria:

(1) One of the friends is female, and the other is male

(2) Their age difference is within “yearDiff”

(3) They are not friends with one another

(4) They should be tagged together in at least one photo

You should return up to n “match pairs.” If there are more than n match pairs, you should break ties as follows:

(1) First choose the pairs with the largest number of shared photos

(2) If there are still ties, choose the pair with the smaller user_id for the female

(3) If there are still ties, choose the pair with the smaller user_id for the male

Query6 (10 points): Suggest friends based on shared friends

For this task, you will suggest friends based on shared friends. In particular, you will find the top n pairs of users in the database who share the most common friends, but who are not friends themselves. Your output will consist of a set of pairs (user1_id, user2_id). No pair should appear in the result set twice; you should always order the pairs so that user1_id < user2_id.

If there are ties, you should give priority to the pair with the smaller user1_id. If there are still ties, then give priority to the pair with the smaller user2_id.

Finally, please note that the _FRIENDS table only records binary friend relationships once, where user1_id is always smaller than user2_id. That is, if users 11 and 12 are friends, the pair (11,12) will appear in the _FRIENDS table, but the pair (12,11) will not appear.

Query7 (10 points): Find oldest and youngest friends

Given the user_id of a user, your task is to find the oldest and youngest friends of that user. If two friends are exactly the same age, meaning that they were born on the same day, month, and year, then you should assume that the friend with the larger user_id is older.

Query8 (10 points): Find the most popular cities to hold events

Find the name of the city with the most events, as well as the number of events in that city. If there is a tie, return the names of all the tied cities. Again, you will place your result in the provided data structures, as demonstrated in MyFakebookOracle.java.

Query9 (10 points): Find the pairs of potential Siblings

A pair of users is potential sibling if they have the same last name and hometown, if they are friends, and if they are less than 10 years apart in age. While doing this, you should compute the year wise difference and not worry about month or day. Pairs of siblings are returned with the lower user_id user first on the line. They are ordered based on the first user_id and in the event of a tie, the second user_id.

4. Compiling and running your code

We provide you an Oracle JDBC Driver (ojdbc6.jar). This driver has been tested with Java JDK 1.6. It may work with later versions of Java as well.

We also suggest that you develop your code in Eclipse (or another IDE of your choice). You can do this by creating a package called ‘project2’ inside Eclipse, and then adding the 3 Java source files to the package.

You should also add your JDBC driver's JAR to Eclipse's class path. In Eclipse, go to 'Project Settings' then 'Java Build Path', and then click on the 'Libraries' tab, then 'Add External JAR'.

If you prefer, you can just use an editor (e.g. emacs) to develop your code. In this case, you should create a directory called 'project2' and put the three Java source files provided to you in this directory.

To compile your code you should change to the directory that contains 'project2'. In other words, suppose you created the directory 'project2' in /your/home/.

cd /your/home/

Then, you can compile the Java source files as follows:

```
javac project2/FakebookOracle.java project2/MyFakebookOracle.java
project2/TestFakebookOracle.java
```

You can run your program as follows (note that you should set the class path (-cp) for ojdbc6.jar):

```
java -cp "/path/to/ojdbc6.jar:" project2/TestFakebookOracle
```

(Use `java -mx64M -cp "/path/to/ojdbc6.jar:" project2/TestFakebookOracle` when compiling on CAEN machines).

/path/to/ is the path to the directory where your ojdbc6.jar file is. Also note the colon (:) after ojdbc6.jar.

If you get a timeout error from Oracle, make sure you are connected from the campus or are using a VPN to connect to the UM (see this page for information: http://www.itcom.itd.umich.edu/vpn/. It is possible that the guest wireless network may not work for remote access to the database without being on the VPN. Or use UM Wireless or a CAEN machine directly for your development.

5. Testing

A good strategy to write embedded SQL is to first test your SQL statements in a more interactive database client such as SLQPlus before putting them inside Java, especially for very complex SQL queries. You have the public data set to test your application. We provide you the output from our official solution querying against the public data (solution-public.txt). You can compare your output with our output to debug. During grading, we run your code on a second (hidden) database.

6. Submission and Grading

You only need to turn in your code via CTools. Submit the following:

- MyFakebookOracle.java --- This should contain all of your code for queries 0-9, and should run with an unmodified version of FakebookOracle.java
- A copy of MyFakebookOracle.java that is named as one of the following:
    - Uniquename_MyFakebookOracle.java
    - Uniquename1_uniquename2_MyFakebookOracle.java

  When you copy MyFakebookOracle.java, do not change the contents of the file.

Grading

We will be grading your answers to queries 1-9 automatically, so it is important that you adhere to the given instructions, and that your file MyFakebookOracle.java works correctly with an unmodified version of FakebookOracle.java. This portion of the project is worth 90 points.

The remaining 10 points will be reserved for Java/SQL programming style. Here are the key elements:

- For each of these tasks, think carefully about what computation should be done in SQL, and what computation should be done in Java. Of course, you can always compute the "correct" answer by fetching all of the data from Oracle, and loading it into Java virtual memory, but this is very inefficient! Instead, **most of the computation can (and should!) be done in SQL, and you should only retrieve the minimum amount of data necessary**.
- Close all SQL resources cleanly. This includes connections, statements, and resultsets. We have posted some hints on how to do this on CTools/Resources/Projects.
- General readability of your queries. Explain the logic briefly in comments if the query is complicated.