EECS 484 Fall 2017: Project 2
Querying the Fakebook Database with JDBC

Due on October 13, 2017 by 11:55PM

Late day policy refer to Grading policies at the end of the spec

Overview

While Project 1 focused primarily on database design, in this project you will focus on writing SQL queries. In addition, you will embed your SQL queries into Java source code (using JDBC) to implement "Fakebook Oracle", a standalone program that answers several queries about the Fakebook database. For this project, you will use a standardized schema that we provide to you, rather than the schema that you designed in Project 1. You will also have access to our public Fakebook dataset for testing.

1. Files Provided to You

You are provided with 4 Java files: `TestFakebookOracle.java`, `FakebookOracle.java`, `TestFakebookOracleFull.java` `MyFakebookOracle.java`. We have also provided a jar file `ojdbc6.jar`. Place all these 4 files, including the jar file under a folder project2. In addition, we have provided you a file `solution-public.txt` showing sample query results. When submitting your completed project, you **only need to** turn in `MyFakebookOracle.java` on AG and `project2.zip` on Canvas. Please check Section 6 for the submission guidelines. We also provide you with a `TestFakebookOracleFull.java` which you can compile and test the time your program took for each query.

1) TestFakebookOracle.java

This file provides the main function for running the program. You can use it to run and test your program, but you don't need to turn it in.

Please only modify the `oracleUserName` and `password` 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, although you are welcome to look at the contents if you are curious.

This class defines the query functions (discussed below in Section 3) as abstract functions, which you must implement for Project 2. It also defines some useful data structures and provides formatted print functions for you to make use of.

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 the query functions. DO NOT make any other changes.

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 output formatting. In the base class `FakebookOracle.java`, a set of print functions have been provided for you to view the query results.

The `MyFakebookOracle` class contains parameterized names for the tables you will need to use in your queries, and they are constructed in the class constructor as shown 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.

```java
// 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";
}
```

For creating and managing the database connection, you should use the predefined object oracleConnection.

4) solution-public.txt

This file contains the output query results from running our official solution implementation on the public dataset provided to you. You can make use of this to check whether or not your queries are generating the same results from the same input dataset.

Note that **your submission will be graded using a different input dataset, so producing correct results on the public dataset is not a guarantee that your solution is entirely correct.** Make sure that your queries are designed to work more generally on any valid input dataset, not just the sample data we provide. Also, 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.

5) TestFakebookOracleFull.java

This file provides another main function for running the program. You can use it to run and test your program, but you don't need to turn it in.

Please only modify the `oracleUserName` and `password` 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) {
```

This program will tell you how much time your program take to run each query. Using too much time in some of the queries will lead to deduction of your points.

2. Tables

For this project, your schema will consist of the following 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

To Access Public Fakebook Data:

<DataType> should be replaced with "`PUBLIC`" to access the public Fakebook data tables.

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

3. Queries (100 points)

There are 10 total queries (Query 0 to Query 9). Query 0 is provided to you as an example, and you are left to implement the remaining nine. The points are as shown. The queries vary tremendously in terms of difficulty. If you get stuck on a harder query, try an easier one first and come back to the tough one later. For all of 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 of these queries, when feasible, you should try to do **most of heavy-lifting to answer the query within SQL**. For example, if a query requires you to present the data in sorted order, use `ORDER BY` in your query rather than retrieving the result and then sorting it within Java.

Also, the grading program we use does impose a **time limit** on the time it waits for a query. If a query appears to be taking too much time, you should consider rewriting it in a different way to make it faster. As mentioned in class, nested queries are usually more expensive to run since most DBMSs do not handle them efficiently.

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

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 users were born and the month in which the fewest users were born. The names of these users 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 `ResultSet` object, and stores the necessary results in a Java object. Finally, it closes both the `Statement` and the `ResultSet` objects.

Query 1: Find information about names (10 points)

The next query asks you to find information about users' names, including (1) the longest first name, (2) the shortest first name, and (3) the most common first name. If there are ties, you should include all of the matches in your result, alphabetically ordered.

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.

```
121    @Override
122    // ***** Query 1 *****
123    // Find information about friend names:
124    // (1) The longest first name (if there is a tie, include all in result)
125    // (2) The shortest first name (if there is a tie, include all in result)
126    // (3) The most common first name, and the number of times it appears (if there
127    //      is a tie, include all in result)
128    //
129    public void findNameInfo() throws SQLException { // Query1
130        // Find the following information from your database and store the information as shown
131        this.longestFirstNames.add("JohnJacobJingleheimerSchmidt");
132        this.shortestFirstNames.add("Al");
133        this.shortestFirstNames.add("Jo");
134        this.shortestFirstNames.add("Bo");
135        this.mostCommonFirstNames.add("John");
136        this.mostCommonFirstNames.add("Jane");
137        this.mostCommonFirstNamesCount = 10;
138    }
```

Query 2: Find "lonely" users (10 points)

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

Query 3: Find "world travelers" (10 points)

The next query asks you to find information about all users who no longer live in their hometowns. In other words, the `current_city` associated with these users should NOT be the same as their `hometown_city` (neither should be `null`). You will place your result (ordered by `user_id`) into the provided data structures.

Query 4: Find information about photo tags (10 points)

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 data types are `VARCHAR`s (for instance, "10" will be less than "2").

Query 5: Suggest friends based on photo tags (15 points)

For this task, you will suggest potential friends to a user. Specifically, you will find the top n pairs of users according to the following criteria:

(1) Both users should be of the same gender
(2) They should be tagged together in at least one photo (They do not have to be friends of the same person)
(3) Their age difference is `<= yearDiff` (just compare the years of birth for this)
(4) They are not friends with one another

You should return up to `n` recommended friend pairs. 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 more than `n` top 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 `user1_id`
(3) If there are still ties, choose the pair with the smaller `user2_id`

Query 6: Suggest friends based on mutual friends (15 points)

For this part, you will suggest potential friends to a user based on mutual friends. In particular, you will find the top `n` pairs of users in the database who have the most common friends, but 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.

Specs clarification: The dataset we will test on will have sufficient qualifying pairs to be n or higher on our test inputs. You will not have to consider situations where the pair of users in the top n have 0 common friends.

Query 7: Find the most popular states to hold events (10 points)

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

Query 8: Find oldest and youngest friends (10 points)

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.

Query 9: Find the pairs of potential siblings (10 points)

A pair of users are potential siblings 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 months or days. Pairs of siblings are returned with the lower `user_id` user first. 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

You are provided with an Oracle JDBC Driver (ojdbc6.jar). This driver has been tested with Java JDK 1.7 and JDK 1.8.

If you are unsure which Java development environment you prefer to use, we suggest that you develop your code in Eclipse. You can do this by creating a Java Project called ‘project2’ inside Eclipse, and then Importing the 3 Java source files to the project.

You should also add your JDBC driver's JAR to Eclipse's classpath. To do this 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., `vi` or `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/Private/EECS484`.

```
cd /your/home/Private/EECS484
```

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

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

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

```
java -Xmx64M -cp "project2/ojdbc6.jar:" project2/TestFakebookOracle
```

Note the colon (:) after ojdbc6.jar.

You can also run the following command to test the time it takes for each of your query.

```
java -Xmx64M -cp "project2/ojdbc6.jar:"
project2/TestFakebookOracleFull
```

NOTE: The time measured by the TestFakebookOracleFull.java is **NOT** final. The time autograder measures **may differ** from the result on your local machine/CAEN. We will be grading based on our autograder result.

Connect from Campus or over a VPN

If you get a timeout error from Oracle, make sure you connect from campus or use the University VPN to connect (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. Alternatively, 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 SQL*Plus before writing them inside Java code, especially for very complex SQL queries. You have the public dataset available to test your application. We provide you with the output from our official solution querying against the public data (available in `solution-public.txt`). You can compare your output with ours to debug. During grading, we will run your code on a second (hidden) dataset.

6. Submission and Grading

To submit on AG, join a team first and then submit only your modified version of `MyFakebookOracle.java`

To submit on Canvas, you will need submit a ZIP file named `project2.zip`

Ensure that the ZIP file contains all of the following files and no others:

1. `MyFakebookOracle.java`
2. `partner.txt` that has one line contains the unique name of your project partner only

Each team **must** submit on **both** platforms.

If you are working in a team, then you should:

1. ***join a team first (on grading system) and***
2. ***Submit only*** `MyFakebookOracle.java` ***on AG.***
3. ***Submit*** `project2.zip` ***on Canvas***

Only one submission per group in AG/Canvas is allowed. *Failure to follow the submission instructions will lead to a penalty of 5 points.*

To create the ZIP file, transfer all of your files to a CAEN Linux machine and execute the following

bash command:

```
% zip -r project2.zip MyFakebookOracle.java partner.txt
```

While no online autograder is currently available, we will be grading your answers to the queries using an automated script, 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`.

We will also grade your code based on the runtime. (See below for details.)

We might later adjust (reduce) the score from running the automated script, if we notice poor Java/SQL programming style. Here are the key elements:

- For each of these tasks, think carefully about what computation is best done in SQL, and what computation is best 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. See Appendix below.
- Make sure your queries are nicely formatted and readable. Explain the logic briefly in comments if the query is complicated.
- You are not being evaluated on optimizing the queries. So, no need to worry about that. However, if you find that they are taking inordinately long, then you should think about simplifying the queries. Else, they could fail the tests.
- Generally, non-nested queries are preferred over nested ones, when feasible. It may not be feasible to do so in all cases. Basically, think about simplifying the queries and use comments to explain, if needed, so that someone other than you can understand your logic.

Grading policies:

Late day policy:

Project 2 is due on Friday, October 13, 2017 at 11:55pm EST. If you do not turn in your

project by this date, or you are unhappy with your work, you may resubmit until Tuesday, October
17, 2017 at 11:55pm EST (4 days after the due date). Each late day (or part thereof) on which you submit incurs a 1% deduction to your **overall course grade** at the end of the semester. These deductions are cumulative for all course projects, and the first two deductions (overall, not per project) will be reprieved.

- If you **fail to join your group before submitting to the grading system**, we will assume that you are submitting on your own and any resemblance to others' solutions will be considered ***violation of the honor code***, which will be reported, and you will ***get 0 points*** in the project. Also, make sure that your Canvas group and your grading system group match.
- If you **submit only to the grading system, but not to Canvas**, you will ***lose 5 points***.
- There is **no need to keep uploading your file to the grading system**. **However**, if you want to upload it multiple times, please keep in mind that there is a **limit of 2 times per day per group**. Keep that in mind so that you do not violate the rule on the last day, when you are most likely to submit the final version of your project. Violation of this rule will lead to obtaining ***50% of the project points***.
- **Runtime (details as below)**

All queries other than query 6

For all queries except query 6, we are enforcing a 1 second time limit on the time taken to run your query and the print function that follows it. Like for the above tests, the result output for your query will not write to screen, in order to avoid any overhead that might impose. You can see from the above times that this is a generous window in which you can receive full credit for your solution.

Since each query will be run twice, once against the public dataset and once against a second, hidden dataset, each of these two runs is worth 5 points (summing to 10 points for each individual query function). For every second by which you exceed this limit, one point will be deducted from your score for any given run. So for example, if, say, query 3 is being run

against the public dataset, these would be the possible scores for when a correct output is produced:

5 pts: <=1 second,
4 pts: <= 2 seconds,
3 pts: <= 3 seconds,
2 pts: <= 4 seconds,
1 pt: <= 5 seconds
0 pts: > 5 seconds

Note that you will not lose more than 5 points per run (no negative scores).

Query 6 policy

For query 6, the same rules apply, but the increment is 15 seconds. This was chosen to be a substantial multiple of the sample solution, and should be very doable. So, for example, each of your runs against the hidden and public datasets will get you one of the following scores for a correct output:

5 pts: <=15 seconds,
4 pts: <= 30 seconds,
3 pts: <= 45 seconds,
2 pts: <= 60 seconds,
1 pt: <= 75 seconds (1.25 minutes)
0 pts: > 75 seconds

If you wish to check your current results for timing on the public data set, use the file `TestFakebookOracleFull.java`, in the project2 folder along with the other Java source files. For timing information, use this class instead of the original TestFakebookOracle to run your queries. Note that seeing a certain time with this test program does not guarantee that time will be measured during grading, it is only a guide. But if you are seeing good time results, that's definitely a good sign, and if they're bad, you may want to reconsider parts of your solution to the underperforming queries.

Checklist for this project:

1. Join a team on the grading system (before making any submissions)
2. Submit `project2.zip` on the grading system
3. Submit `project2.zip` on Canvas
4. Do not upload your file to the grading system more than twice per day

Appendix: Tips on Closing Your JDBC Resources Cleanly

It is important that you close your JDBC connections properly. Otherwise, you risk getting locked out of the database server. Even if you kill your Java program, it is possible that the database server thinks that the client program is still around. It may keep its end of the connection open, waiting for data. This could eventually lock you out of the database if you end up creating too many instances of open connections.

Here is a real example of an unfortunate situation posted by a student:

"So I've been having this same problem, but I locked myself over 24 hours ago and I still don't have access. I'm not sure what to do since ITS isn't open and if CAEN can't help I'm skeptical anyway. I created this problem by my SQL queries were running on forever (can you even get into an infinite loop in SQL? I think I was) and when you Ctrl+C out it closes SQL improperly. I've learned from my mistake now, but it's too late and I can't get back in."

Here are a few tips to help reduce the likelihood of the above type of problem:

1. First, use SQL*Plus to debug your queries rather than using Java. Also make sure you quit your SQL*Plus sessions, otherwise you can still get locked out. It may help to design your queries on paper first and avoid too much nesting of queries. Nested queries tend to run slower as query optimizers have difficulty handling them. The above student was smart enough to do that but still ran into trouble. So, let's look at one more thing to do (Step 2) that may help.
2. Make sure you close all `Connection`s, `Statement`s, and `ResultSet`s in your code. This is tricky to do. Read this for some of the nuances:

   http://blog.shinetech.com/2007/08/04/howtoclosejdbcresourcesproperlyeverytime/

   The problem is, even closing a connection can lead to an exception in rare cases.

We suggest using try-with-resources statements in your Java code to automatically close any JDBC objects that you create. This is a newer feature introduced in Java SE 7 that makes closing resources easier and more reliable. The next page shows an example of how this can be done for JDBC connections.

```
// Example use of a try-with-resources statement
public static void viewTable(Connection con) throws SQLException {

    String query = "SELECT COF_NAME, SUP_ID, PRICE, SALES, TOTAL FROM
COFFEES";

    try (Statement stmt = con.createStatement()) {
        ResultSet rs = stmt.executeQuery(query);

        while (rs.next()) {
            String coffeeName = rs.getString("COF_NAME");
            int supplierID = rs.getInt("SUP_ID");
            float price = rs.getFloat("PRICE");
            int sales = rs.getInt("SALES");
            int total = rs.getInt("TOTAL");

            System.out.println(coffeeName + ", " + supplierID + ", " +
                               price + ", " + sales + ", " + total);
        }
    } catch (SQLException e) {
        System.err.println(e.getMessage());
    }
}
```

This code is adapted from an example at:

https://docs.oracle.com/javase/tutorial/essential/exceptions/tryResourceClose.html

Feel free to take the code and adapt it to your needs. The above code assumes that you only need to use a single `Statement` object, for example. That may not always be appropriate. For example, you may sometimes want to use multiple `Statement` objects to execute different fixed queries, in which case you could have one try-with-resources block for each `Statement` object that you create. Since a `Connection` object is provided to you in the Project 2 code, you do not need to worry about creating a try-with-resources block for the `Connection` object that you use in your query implementations.

If you think about the problem, it is actually pretty hard for a database server to distinguish between a slow client at the other end and a dead client. Remember that the communication between the client and the server occurs over a network. Doing all you can within Java to close the connections in all possible situations (including when queries fail) will help the server greatly.