# EECS 484 Fall 2015: Project 2
# Querying the Fakebook Database with JDBC

### Due on October 15, 2015 by 11:55PM

## 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 3 Java files: `TestFakebookOracle.java`, `FakebookOracle.java` and `MyFakebookOracle.java`, in addition to a file showing sample query results. When submitting your completed project, you only need to turn in `MyFakebookOracle.java`.

### 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, 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.

```
10 public class MyFakebookOracle extends FakebookOracle {
11
12     static String prefix = "crowella.";
```

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.

```
// 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.

## 2. 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

**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 (`crowella`). 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 `crowella.PUBLIC_USERS`. In the Java files provided to you, the above table names are already pre-configured in the given code.

## 3. Queries (90 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 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 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, you should try to do the computational work within SQL as much as is possible. 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 (for example, several minutes) you should consider rewriting it in a different way to make it faster. Nested queries are usually more expensive to run.

## 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 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 `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 user 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.

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” friends (10 points)

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.

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

The next query asks you to find information about all friends 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 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: Find friends to set up on dates (10 points)

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

## Query 6: Suggest friends based on shared friends (10 points)

For this part, you will suggest potential friends to a user 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.

### 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. It may also work with later versions of Java.

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/`.

```
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 your copy of 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 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

You only need to turn in your code via Canvas. Please submit only the following:

- `MyFakebookOracle.java` - This should contain all of your code for queries 0-9, and should be able to run with an unmodified version of `FakebookOracle.java`

**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 evaluating your 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. This includes `Connection`s, `Statement`s, and `ResultSet`s. We have posted some hints on how to do this on Canvas in Files/Projects/Project_2_Logistics.pdf.
- 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 a grader can understand your logic.