Discussion 4

More SQL, JDBC, Project 2 Intro

EECS 484

Logistics

- **Homework 2** released, due **Oct 4th** at 11:45 PM ET
- **Project 2** released, due **Oct 22nd** at 11:45 PM ET
- **Midterm: Oct 10th**, 6:15–8:15 PM ET, at **CCCB** on Central Campus
  - Midterm Review Session: Oct 2nd, 6:30–8:00 PM ET, at BBB 1670
  - More details for the midterm are coming soon

More SQL

Group By and Having

- Query Syntax:
  - SELECT attribute_list (5)
    FROM relation_list (1)
    WHERE conditional_statement (2)
    GROUP BY grouping_list (3)
    HAVING group_conditionals (4)
    ORDER BY attribute_list (6)
  - Think of GROUP BY as defining the groups and HAVING as a WHERE clause for the groups

(1) Gather data from relation_list
(2) Filter out unwanted data
(3) Form groups
(4) Filter out unwanted groups
(5) Select desired attributes and evaluate aggregate functions
(6) Sort

Counting

- We can count the number of rows in a table
  - SELECT COUNT(*) FROM USERS;
- What if we wanted to count occurences in a table?
  - How many people have the name ‘Jane’?
    - SELECT COUNT(*) FROM USERS WHERE name=’Jane’;

Counting

- What if we wanted to count occurences in a table?
  - What if we wanted to get counts for all the names?
    - SELECT COUNT(*), name FROM USERS GROUP BY name;
    - Aggregate function over a group
      - Create groups of rows based on the first name
        - Group of ‘John’s, group of ‘Jane’s, etc.
      - Run the function (in this case COUNT(*)) over all the groups
      - End result, count of each first name

User_ID	Name	Age
1	Jane	22
2	John	32
3	Alice	34
4	John	22
5	Jane	34

COUNT(*)	Name
2	Jane
2	John
1	Alice

Evaluation Order Example

```
SELECT Name, COUNT(*)
FROM Users ……(1)
WHERE Age > 30 ……(2)
GROUP BY Name ……(3)
HAVING COUNT(*) > 1……(4)
ORDER BY COUNT(*) DESC;.....(5)
```

User_ID	Name	Age
1	Jane	32
2	John	32
3	Alice	34
4	John	22
5	Jane	34
6	John	56
7	John	40

Evaluation Order Example

(1) FROM Users

User_ID	Name	Age
1	Jane	32
2	John	32
3	Alice	34
4	John	22
5	Jane	34
6	John	56
7	John	40

(2) WHERE Age > 30

User_ID	Name	Age
1	Jane	32
2	John	32
3	Alice	34
4	John	22
5	Jane	34
6	John	56
7	John	40

Evaluation Order Example

(2) WHERE Age > 30

User_ID	Name	Age
1	Jane	32
2	John	32
3	Alice	34
4	John	22
5	Jane	34
6	John	56
7	John	40

(3) GROUP BY Name

User_ID	Name	Age
1	Jane	32
5	Jane	34
2	John	32
6	John	56
7	John	40
3	Alice	34

Evaluation Order Example

(3) GROUP BY Name

User_ID	Name	Age
1	Jane	32
5	Jane	34
2	John	32
6	John	56
7	John	40
3	Alice	34

COUNT(*) = 2

COUNT(*) = 3

COUNT(*) = 1

(4) HAVING COUNT(*) > 1

User_ID	Name	Age
1	Jane	32
5	Jane	34
2	John	32
6	John	56
7	John	40

Evaluation Order Example

(4) HAVING COUNT(*) > 1

User_ID	Name	Age
1	Jane	32
5	Jane	34
2	John	32
6	John	56
7	John	40

(5) SELECT Name, COUNT(*)

Name	COUNT(*)
Jane	2
John	3

Evaluation Order Example

(5) SELECT Name, COUNT(*)

Name	COUNT(*)
Jane	2
John	3

(6) ORDER BY COUNT(*) DESC

Name	COUNT(*)
John	3
Jane	2

Evaluation Order Example

```
SELECT Name, COUNT(*)
FROM Users
WHERE Age > 30
GROUP BY Name
HAVING COUNT(*) > 1
ORDER BY COUNT(*) DESC;
```

User_ID	Name	Age
1	Jane	32
2	John	32
3	Alice	34
4	John	22
5	Jane	34
6	John	56
7	John	40

Name	COUNT(*)
John	3
Jane	2

Nested Queries

- What if I now want the names of everyone who's at the same age as at least one other person?
  - SELECT name, age
    FROM USERS
    GROUP BY AGE
    HAVING COUNT(*) > 1;
  - **ORA-00979: not a GROUP BY expression**
  - We have groups of people who share the same age
  - People in these groups have different names though

User_ID	Name	Age
1	Jane	22
2	John	32
3	Alice	34
4	John	22
5	Jane	34

Nested Queries

```
SELECT name, age
FROM USERS
GROUP BY AGE
HAVING COUNT(*) > 1;
```

SELECT statement must be a subset of what we've grouped by

- Intuition, we are selecting groups now, not rows
- Groups do not have names in this case, only ages

Solution: nested queries!

User_ID	Name	Age
1	Jane	22
2	John	32
3	Alice	34
4	John	22
5	Jane	34

Nested Queries Cont.

- SELECT u0.name, u0.age
  FROM USERS u0
  WHERE u0.age IN
    (SELECT u1.age
    FROM USERS u1
    GROUP BY u1.age
    HAVING COUNT(*) > 1);
- Unpacking this:
  - Inner query: Select all the user ages such that multiple users share the same age
    - Groups of users sharing the same age
    - Count of entries in group > 1 (at least 2 users share the same age)
  - Outer query: Select all the user names and ages such that the age is one of the ages we found in the inner query

User_ID	Name	Age
1	Jane	22
2	John	32
3	Alice	34
4	John	22
5	Jane	34

Name	Age
Jane	22
Alice	34
John	22
Jane	34

Order By and Rownum

- Rownum allows us to specify how many rows to return
  - Pseudocolumn that begins at 1
- Order By allows us to sort data
  - ORDER BY column_name ASC
    - Default (ORDER BY column_name)
  - ORDER BY column_name DESC

Order By and Rownum

- Example:
- Find the name and the age of the two oldest persons
  - SELECT Name, Age
    FROM (SELECT * FROM USERS
    ORDER BY AGE DESC)
    WHERE ROWNUM <= 2

User_ID	Name	Age
1	Jane	22
2	John	32
3	Alice	34
4	John	22
5	Jane	34

Name	Age
Alice	34
Jane	34

- Orders the USER table by age in descending order, then selects the top 2 rows
  - Need nested query, otherwise we'd take the top 2 rows before sorting by age
- Can break ties further with an additional column in the ORDER BY

Order By and Rownum Cont.

- Queries can get quite messy
  - Combine GROUP BY, HAVING, ORDER BY, and ROWNUM
  - WHERE clause comes before GROUP BY and ORDER BY
  - But if we apply our ROWNUM condition before grouping it will filter out excess rows not groups
  - This means we need nested queries if we want to select a few GROUPS instead of ROWS

User_ID	Name	Age
1	Jane	22
2	John	32
3	Alice	34
4	John	22
5	Jane	34

Name	Age
Alice	34
Jane	34

Order By and Rownum Cont.

- Example
  - SELECT * FROM
    (SELECT u0.name, u0.age
    FROM USERS u0
    WHERE u0.age IN
    (SELECT u1.age FROM USERS u1
    GROUP BY u1.age
    HAVING COUNT(*) > 1)
    ORDER BY u0.name, u0.age)
    WHERE ROWNUM < 3;

User_ID	Name	Age
1	Jane	22
2	John	32
3	Alice	34
4	John	22
5	Jane	34

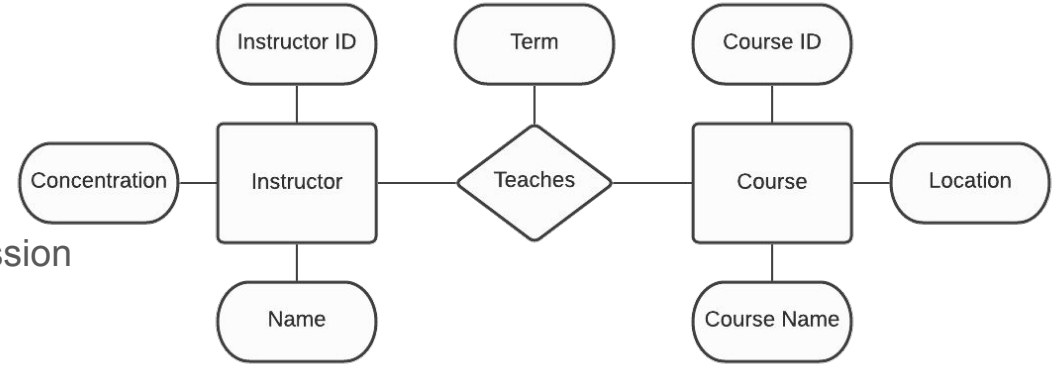
Name	Age
Alice	34
Jane	22

  - Query from before but take the top two people sorted by alphabetical name and then age

Practice Problems :D

Example Problem

- Instructors teach courses
  - Same problem from last discussion



Course ID	Course Name	Location
EECS 575	Crypto	1690BBB
EECS 484	Databases	1670BBB
EECS 482	OS	1690BBB
EECS 388	Security	404BBB
AERO 548	Astrodynamics	FXB1012
EECS 999	Redacted	???

Instructor ID	Name	Concentration
1112223456	Alice	Cryptography
9876543210	Bob	Boring
0000000000	SQL	Databases
0123456789	Eve	Hacking :D
1107201969	Buzz	Space
3333333333	Mr. Meow	Meowing

Instructor ID	Course ID	Term
1112223456	EECS 575	F20
0000000000	EECS 484	F20
9876543210	EECS 482	W19
1107201969	AERO 548	W19
1112223456	EECS 388	W21
0123456789	EECS 388	W21

Q1

- Get the course ID with the most instructors who teach it along with the number of instructors for the course
  - Resolve ties by using the smaller course ID
  - Things you may need to use (Might not need all of them)
    - SELECT
    - FROM
    - WHERE
    - GROUP BY
    - HAVING
    - ORDER BY
    - JOIN

Schema:
Course (Course_ID, Course_Name, Location)
Instructor (Instructor_ID, Name, Concentration)
Teaches (Instructor_ID, Course_ID, Term)

Q1 Answer

Schema:
Course (Course_ID, Course_Name, Location)
Instructor (Instructor_ID, Name, Concentration)
Teaches (Instructor_ID, Course_ID, Term)

- SELECT * FROM (
  SELECT T.Course_ID, COUNT(*)
  FROM Teaches T
  GROUP BY T.Course_ID
  ORDER BY COUNT(*) DESC, T.Course_ID ASC
) WHERE ROWNUM < 2;
- Why do we need two selects?
  - To apply the ROWNUM < 2 after grouping and sorting the data

Q2

- Get the course IDs for all courses that are only taught by one instructor and only in one term. Get the corresponding instructor name for each course as well.
  - Order by smallest course ID first
  - Things you may need to use (Might not need all of them)
    - SELECT
    - FROM
    - WHERE
    - GROUP BY
    - HAVING
    - ORDER BY
    - JOIN

Schema:
Course (Course_ID, Course_Name, Location)
Instructor (Instructor_ID, Name, Concentration)
Teaches (Instructor_ID, Course_ID, Term)

Q2 Answer

- SELECT T0.Course_ID, I.Name
  FROM Teaches T0
  INNER JOIN Instructor I ON T0.Instructor_ID = I.Instructor_ID
  WHERE T0.Course_ID IN (
  SELECT T.Course_ID
  FROM Teaches T
  GROUP BY T.Course_ID
  HAVING COUNT(*) = 1)
  ORDER BY T0.Course_ID;

Schema:
Course (Course_ID, Course_Name, Location)
Instructor (Instructor_ID, Name, Concentration)
Teaches (Instructor_ID, Course_ID, Term)

Java Database Connectivity (JDBC) and Project 2

No Need to Hurry!

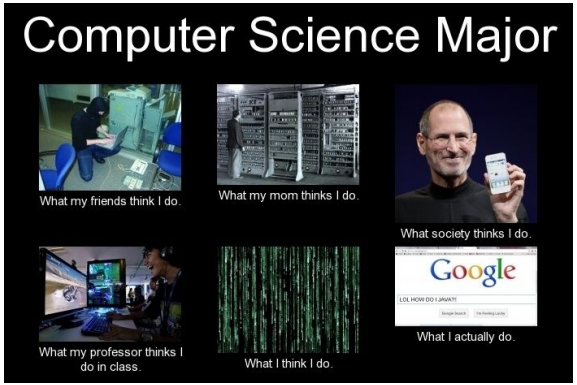
- We release the project 2 for those who are interested to start early!
- Feel free to take a rest after project 1 and start project 2 when you are ready!

JDBC Overview

- Right now all of our queries are running in the terminal
  - Not very useful - need to incorporate into an application
- JDBC is a Java API that allows for database connections
  - Send queries to database and retrieve results
  - Results can be parsed in the application then
- Project 2 makes use of JDBC
  - Requires Java
    - We give you most of the code
    - You will be able to understand much of the syntax if you know C++
  - 9 Queries to complete
  - Efficiency matters - the Autograder checks your runtime
    - Submits make take around 10 minutes to run all test cases (used to be 40 min)

Project 2 Setup

- You will be working with public database tables just like in P1
    - We give you a set of constants for the table names - use those



Project 2 Setup

- File structure of project2 directory
  - PublicFakebookOracleConstants.java
    - List of constants to use for selecting from tables
  - FakebookOracleDataStructures.java
    - Data structures you will use for storing and returning data

Project 2 Setup

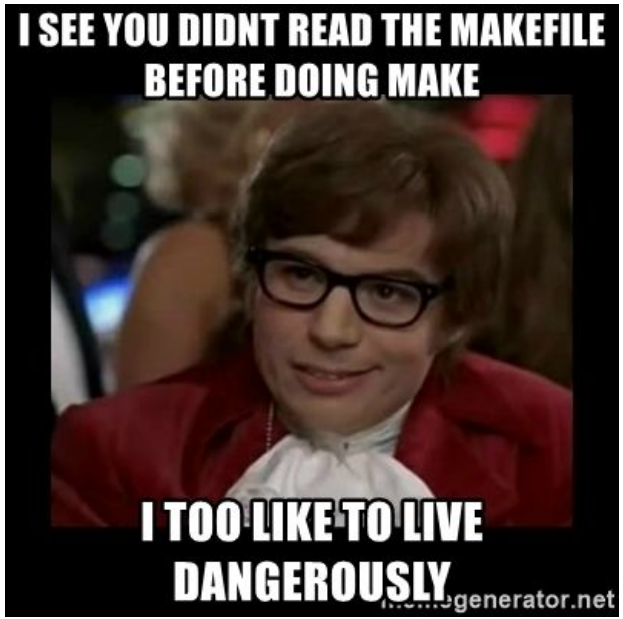
- File structure of project2 directory
  - FakebookOracleMain.java
    - Main function that controls program
    - You will need to edit this to insert your Oracle username and password
      - Because this is stored in plaintext, we recommend you set your Oracle password to a password you don't use anywhere else
      - Only used for testing locally
  - StudentFakebookOracle.java
    - Only file you submit. It will have all your queries

Project 2 Code Structure

- File Structure
    - project2
        - FakebookOracle.java
        - FakebookOracleDataStructures.java
        - FakebookOracleMain.java
        - FakebookOracleUtilities.java
        - PublicFakebookOracleConstants.java
        - StudentFakebookOracle.java
        - ojdbc6.jar - Jar file for the JDBC libraries
    - Makefile
    - PublicTime.txt - contains times each query should take to execute
        - Don’t be worried if you’re a little over these times (Autograder is a little lenient)
    - PublicSolution.txt - Correct answers for each query
    - README.md

Running the Java Code

- We give you the Makefile and all necessary files to run the code
- n the root directory (with the Makefile) you can run Make commands
  - make query-all
    - See all query outputs
  - make time-all
    - See the the timings from all query outputs
  - make query[0-9]
    - Runs the query with the specified number
  - make time[0-9]
    - Times the query with the specific number



Query 0 Walkthrough Step 0

- Query 0 is provided for you as an example on these queries
  - Finds the following data:
    - Month with most users born
    - Month with fewest users born
    - Name of all users born in the above two months

```
public BirthMonthInfo findMonthOfBirthInfo() throws SQLException {
    try (Statement stmt = oracle.createStatement(FakebookOracleConstants.AllScroll, FakebookOracleConstants.ReadOnly)) {
        // Step 1
        // -----------
        // * Find the total number of users with birth month info
        // * Find the month in which the most users were born
        // * Find the month in which the fewest (but at least 1) users were born
        ResultSet rst = stmt.executeQuery(
            "SELECT COUNT(*) AS Birthed, Month_of_Birth " +          // select birth months and number of uses with that birth month
            "FROM " + UsersTable + " " +                             // from all users
            "WHERE Month_of_Birth IS NOT NULL " +                    // for which a birth month is available
            "GROUP BY Month_of_Birth " +                             // group into buckets by birth month
            "ORDER BY Birthed DESC, Month_of_Birth ASC");            // sort by users born in that month, descending; break ties by birth month

        int mostMonth = 0;
        int leastMonth = 0;
        int total = 0;
        while (rst.next()) {                    // step through result rows/records one by one
            if (rst.isFirst()) {                // if first record
                mostMonth = rst.getInt(2);      //   it is the month with the most
            }
            if (rst.isLast()) {                 // if last record
                leastMonth = rst.getInt(2);     //   it is the month with the least
            }
            total += rst.getInt(1);             // get the first field's value as an integer
        }
        BirthMonthInfo info = new BirthMonthInfo(total, mostMonth, leastMonth);
```

- Start with creating a statement
  - Reads constants in and establishes a connection with the db
  - Surround with try catch in case the statement connection throws an error

```java
public BirthMonthInfo findMonthOfBirthInfo() throws SQLException {
    try (Statement stmt = oracle.createStatement(FakebookOracleConstants.AllScroll, FakebookOracleConstants.ReadOnly)) {
        // Step 1
        // ------------
        // * Find the total number of users with birth month info
        // * Find the month in which the most users were born
        // * Find the month in which the fewest (but at least 1) users were born
        ResultSet rst = stmt.executeQuery(
            "SELECT COUNT(*) AS Birthed, Month_of_Birth " +          // select birth months and number of uses with that birth month
            "FROM " + UsersTable + " " +                             // from all users
            "WHERE Month_of_Birth IS NOT NULL " +                    // for which a birth month is available
            "GROUP BY Month_of_Birth " +                             // group into buckets by birth month
            "ORDER BY Birthed DESC, Month_of_Birth ASC");            // sort by users born in that month, descending; break ties by birth month

        int mostMonth = 0;
        int leastMonth = 0;
        int total = 0;
        while (rst.next()) {                        // step through result rows/records one by one
            if (rst.isFirst()) {                    // if first record
                mostMonth = rst.getInt(2);          //   it is the month with the most
            }
            if (rst.isLast()) {                     // if last record
                leastMonth = rst.getInt(2);         //   it is the month with the least
            }
            total += rst.getInt(1);                 // get the first field's value as an integer
        }
        BirthMonthInfo info = new BirthMonthInfo(total, mostMonth, leastMonth);
```

Query 0 Walkthrough Step 1

- Count the number of users in each month
  - Done by the group by statement
  - Order by the count of users in each month - most users first
    - Break ties based on alphabetical order of birth month

```
public BirthMonthInfo findMonthOfBirthInfo() throws SQLException {
    try (Statement stmt = oracle.createStatement(FakebookOracleConstants.AllScroll, FakebookOracleConstants.ReadOnly)) {
        // Step 1
        // -----------
        // * Find the total number of users with birth month info
        // * Find the month in which the most users were born
        // * Find the month in which the fewest (but at least 1) users were born
        ResultSet rst = stmt.executeQuery(
            "SELECT COUNT(*) AS Birthed, Month_of_Birth " +          // select birth months and number of uses with that birth month
            "FROM " + UsersTable + " " +                             // from all users
            "WHERE Month_of_Birth IS NOT NULL " +                    // for which a birth month is available
            "GROUP BY Month_of_Birth " +                             // group into buckets by birth month
            "ORDER BY Birthed DESC, Month_of_Birth ASC");            // sort by users born in that month, descending; break ties by birth month

        int mostMonth = 0;
        int leastMonth = 0;
        int total = 0;
        while (rst.next()) {                    // step through result rows/records one by one
            if (rst.isFirst()) {                // if first record
                mostMonth = rst.getInt(2);      //   it is the month with the most
            }
            if (rst.isLast()) {                 // if last record
                leastMonth = rst.getInt(2);     //   it is the month with the least
            }
            total += rst.getInt(1);             // get the first field's value as an integer
        }
        BirthMonthInfo info = new BirthMonthInfo(total, mostMonth, leastMonth);
```

```java
public BirthMonthInfo findMonthOfBirthInfo() throws SQLException {
    try (Statement stmt = oracle.createStatement(FakebookOracleConstants.AllScroll, FakebookOracleConstants.ReadOnly)) {
        // Step 1
        // -----------
        // * Find the total number of users with birth month info
        // * Find the month in which the most users were born
        // * Find the month in which the fewest (but at least 1) users were born
        ResultSet rst = stmt.executeQuery(
            "SELECT COUNT(*) AS Birthed, Month_of_Birth " +          // select birth months and number of uses with that birth month
            "FROM " + UsersTable + " " +                             // from all users
            "WHERE Month_of_Birth IS NOT NULL " +                    // for which a birth month is available
            "GROUP BY Month_of_Birth " +                             // group into buckets by birth month
            "ORDER BY Birthed DESC, Month_of_Birth ASC");            // sort by users born in that month, descending; break ties by birth month

        int mostMonth = 0;
        int leastMonth = 0;
        int total = 0;
        while (rst.next()) {                        // step through result rows/records one by one
            if (rst.isFirst()) {                    // if first record
                mostMonth = rst.getInt(2);          //   it is the month with the most
            }
            if (rst.isLast()) {                     // if last record
                leastMonth = rst.getInt(2);         //   it is the month with the least
            }
            total += rst.getInt(1);                 // get the first field's value as an integer
        }
        BirthMonthInfo info = new BirthMonthInfo(total, mostMonth, leastMonth);
```

Query 0 Walkthrough Step 1 Cont'd

- Go through the results in Java
- While rst.next()
  - Goes through the result set returned one row at a time
  - rst.isFirst() is true for the first row
    - Get the month with the most users born in it
    - getInt(2) returns the second column in the current row and parses it as an int
      - Other functions for different data types
      - 1 Indexing as opposed to 0!
  - rst.isLast() is true for the last row
    - Get the month with the least users born in it
- Add information found to a brand new data structure

```
public BirthMonthInfo findMonthOfBirthInfo() throws SQLException {
    try (Statement stmt = oracle.createStatement(FakebookOracleConstants.AllScroll, FakebookOracleConstants.ReadOnly)) {
        // Step 1
        // -----------
        // * Find the total number of users with birth month info
        // * Find the month in which the most users were born
        // * Find the month in which the fewest (but at least 1) users were born
        ResultSet rst = stmt.executeQuery(
            "SELECT COUNT(*) AS Birthed, Month_of_Birth " +          // select birth months and number of uses with that birth month
            "FROM " + UsersTable + " " +                             // from all users
            "WHERE Month_of_Birth IS NOT NULL " +                    // for which a birth month is available
            "GROUP BY Month_of_Birth " +                             // group into buckets by birth month
            "ORDER BY Birthed DESC, Month_of_Birth ASC");            // sort by users born in that month, descending; break ties by birth month

        int mostMonth = 0;
        int leastMonth = 0;
        int total = 0;
        while (rst.next()) {                    // step through result rows/records one by one
            if (rst.isFirst()) {                // if first record
                mostMonth = rst.getInt(2);      //   it is the month with the most
            }
            if (rst.isLast()) {                 // if last record
                leastMonth = rst.getInt(2);     //   it is the month with the least
            }
            total += rst.getInt(1);             // get the first field's value as an integer
        }
        BirthMonthInfo info = new BirthMonthInfo(total, mostMonth, leastMonth);
```

"Arrays start at 0!"
ResultSet in JDBC:

```java
public BirthMonthInfo findMonthOfBirthInfo() throws SQLException {
    try (Statement stmt = oracle.createStatement(FakebookOracleConstants.AllScroll, FakebookOracleConstants.ReadOnly)) {
        // Step 1
        // -----------
        // * Find the total number of users with birth month info
        // * Find the month in which the most users were born
        // * Find the month in which the fewest (but at least 1) users were born
        ResultSet rst = stmt.executeQuery(
            "SELECT COUNT(*) AS Birthed, Month_of_Birth " +        // select birth months and number of uses with that birth month
            "FROM " + UsersTable + " " +                          // from all users
            "WHERE Month_of_Birth IS NOT NULL " +                 // for which a birth month is available
            "GROUP BY Month_of_Birth " +                          // group into buckets by birth month
            "ORDER BY Birthed DESC, Month_of_Birth ASC");         // sort by users born in that month, descending; break ties by birth month

        int mostMonth = 0;
        int leastMonth = 0;
        int total = 0;
        while (rst.next()) {                    // step through result rows/records one by one
            if (rst.isFirst()) {                // if first record
                mostMonth = rst.getInt(2);      //   it is the month with the most
            }
            if (rst.isLast()) {                 // if last record
                leastMonth = rst.getInt(2);     //   it is the month with the least
            }
            total += rst.getInt(1);             // get the first field's value as an integer
        }
        BirthMonthInfo info = new BirthMonthInfo(total, mostMonth, leastMonth);
```

Query 0 Walkthrough Step 2

- Execute another query
  - Reassigning rst closes the previous
    resultset (overwriting)
  - Insert data from the last query result
    into the next query
    - mostMonth is used in the WHERE clause to find all the users born in the month with most births
- Read through the other result set
  - Add each user's information to the data structure to return

```
// Step 2
// ------------
// * Get the names of users born in the most popular birth month
rst = stmt.executeQuery(
    "SELECT User_ID, First_Name, Last_Name " +              // select ID, first name, and last name
    "FROM " + UsersTable + " " +                           // from all users
    "WHERE Month_of_Birth = " + mostMonth + " " +          // born in the most popular birth month
    "ORDER BY User_ID");                                   // sort smaller IDs first

while (rst.next()) {
    info.addMostPopularBirthMonthUser(new UserInfo(rst.getLong(1), rst.getString(2), rst.getString(3)));
}

// Step 3
// ------------
// * Get the names of users born in the least popular birth month
rst = stmt.executeQuery(
    "SELECT User_ID, First_Name, Last_Name " +              // select ID, first name, and last name
    "FROM " + UsersTable + " " +                           // from all users
    "WHERE Month_of_Birth = " + leastMonth + " " +         // born in the least popular birth month
    "ORDER BY User_ID");                                   // sort smaller IDs first

while (rst.next()) {
    info.addLeastPopularBirthMonthUser(new UserInfo(rst.getLong(1), rst.getString(2), rst.getString(3)));
}

// Step 4
// ------------
// * Close resources being used
rst.close();
stmt.close();                       // if you close the statement first, the result set gets closed automatically

return info;
```

```
// Step 2
// -------------
// * Get the names of users born in the most popular birth month
rst = stmt.executeQuery(
    "SELECT User_ID, First_Name, Last_Name " +                  // select ID, first name, and last name
    "FROM " + UsersTable + " " +                                // from all users
    "WHERE Month_of_Birth = " + mostMonth + " " +               // born in the most popular birth month
    "ORDER BY User_ID");                                        // sort smaller IDs first

while (rst.next()) {
    info.addMostPopularBirthMonthUser(new UserInfo(rst.getLong(1), rst.getString(2), rst.getString(3)));
}

// Step 3
// -------------
// * Get the names of users born in the least popular birth month
rst = stmt.executeQuery(
    "SELECT User_ID, First_Name, Last_Name " +                  // select ID, first name, and last name
    "FROM " + UsersTable + " " +                                // from all users
    "WHERE Month_of_Birth = " + leastMonth + " " +              // born in the least popular birth month
    "ORDER BY User_ID");                                        // sort smaller IDs first

while (rst.next()) {
    info.addLeastPopularBirthMonthUser(new UserInfo(rst.getLong(1), rst.getString(2), rst.getString(3)));
}

// Step 4
// -------------
// * Close resources being used
rst.close();
stmt.close();                           // if you close the statement first, the result set gets closed automatically

return info;
```

Query 0 Walkthrough Step 3

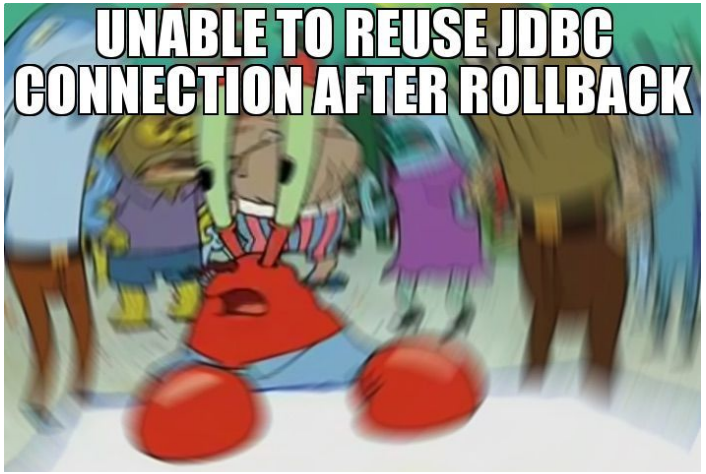
- Execute another query
  - Same as last query except to find the users born in the month with the least number of users born in


```
BirthMonthInfo info = new BirthMonthInfo(total, mostMonth, leastMonth);

// Step 2
// ------------
// * Get the names of users born in the most popular birth month
rst = stmt.executeQuery(
    "SELECT User_ID, First_Name, Last_Name " +              // select ID, first name, and last name
    "FROM " + UsersTable + " " +                           // from all users
    "WHERE Month_of_Birth = " + mostMonth + " " +          // born in the most popular birth month
    "ORDER BY User_ID");                                   // sort smaller IDs first

while (rst.next()) {
    info.addMostPopularBirthMonthUser(new UserInfo(rst.getLong(1), rst.getString(2), rst.getString(3)));
}

// Step 3
// ------------
// * Get the names of users born in the least popular birth month
rst = stmt.executeQuery(
    "SELECT User_ID, First_Name, Last_Name " +              // select ID, first name, and last name
    "FROM " + UsersTable + " " +                           // from all users
    "WHERE Month_of_Birth = " + leastMonth + " " +         // born in the least popular birth month
    "ORDER BY User_ID");                                   // sort smaller IDs first

while (rst.next()) {
    info.addLeastPopularBirthMonthUser(new UserInfo(rst.getLong(1), rst.getString(2), rst.getString(3)));
}

// Step 4
// ------------
// * Close resources being used
rst.close();
stmt.close();                          // if you close the statement first, the result set gets closed automatically

return info;
```

```
BirthMonthInfo info = new BirthMonthInfo(total, mostMonth, leastMonth);

// Step 2
// -------------
// * Get the names of users born in the most popular birth month
rst = stmt.executeQuery(
    "SELECT User_ID, First_Name, Last_Name " +                  // select ID, first name, and last name
    "FROM " + UsersTable + " " +                                // from all users
    "WHERE Month_of_Birth = " + mostMonth + " " +               // born in the most popular birth month
    "ORDER BY User_ID");                                        // sort smaller IDs first

while (rst.next()) {
    info.addMostPopularBirthMonthUser(new UserInfo(rst.getLong(1), rst.getString(2), rst.getString(3)));
}

// Step 3
// -------------
// * Get the names of users born in the least popular birth month
rst = stmt.executeQuery(
    "SELECT User_ID, First_Name, Last_Name " +                  // select ID, first name, and last name
    "FROM " + UsersTable + " " +                                // from all users
    "WHERE Month_of_Birth = " + leastMonth + " " +              // born in the least popular birth month
    "ORDER BY User_ID");                                        // sort smaller IDs first

while (rst.next()) {
    info.addLeastPopularBirthMonthUser(new UserInfo(rst.getLong(1), rst.getString(2), rst.getString(3)));
}

// Step 4
// -------------
// * Close resources being used
rst.close();
stmt.close();                           // if you close the statement first, the result set gets closed automatically

return info;
```

Query 0 Walkthrough Step 4

- Clear out all resources used
  - Close the resultset and statement
    - Important! Can cause issues on the Autograder
- Return the information collected



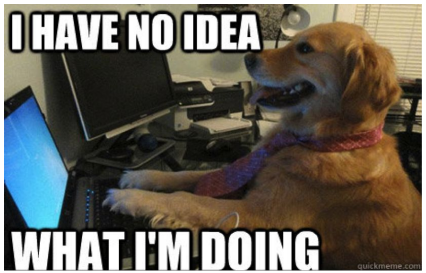

```
    // Step 3
    // ------------
    // * Get the names of users born in the least popular birth month
    rst = stmt.executeQuery(
        "SELECT User_ID, First_Name, Last_Name " +              // select ID, first name, and last name
        "FROM " + UsersTable + " " +                            // from all users
        "WHERE Month_of_Birth = " + leastMonth + " " +          // born in the least popular birth month
        "ORDER BY User_ID");                                    // sort smaller IDs first

    while (rst.next()) {
        info.addLeastPopularBirthMonthUser(new UserInfo(rst.getLong(1), rst.getString(2), rst.getString(3)));
    }

    // Step 4
    // ------------
    // * Close resources being used
    rst.close();
    stmt.close();                           // if you close the statement first, the result set gets closed automatically

    return info;

}
catch (SQLException e) {
    System.err.println(e.getMessage());
    return new BirthMonthInfo(-1, -1, -1);
}
```

```
    // Step 3
    // -------------
    // * Get the names of users born in the least popular birth month
    rst = stmt.executeQuery(
        "SELECT User_ID, First_Name, Last_Name " +              // select ID, first name, and last name
        "FROM " + UsersTable + " " +                            // from all users
        "WHERE Month_of_Birth = " + leastMonth + " " +          // born in the least popular birth month
        "ORDER BY User_ID");                                    // sort smaller IDs first

    while (rst.next()) {
        info.addLeastPopularBirthMonthUser(new UserInfo(rst.getLong(1), rst.getString(2), rst.getString(3)));
    }

    // Step 4
    // -------------
    // * Close resources being used
    rst.close();
    stmt.close();                           // if you close the statement first, the result set gets closed automatically

    return info;

}
catch (SQLException e) {
    System.err.println(e.getMessage());
    return new BirthMonthInfo(-1, -1, -1);
}
```

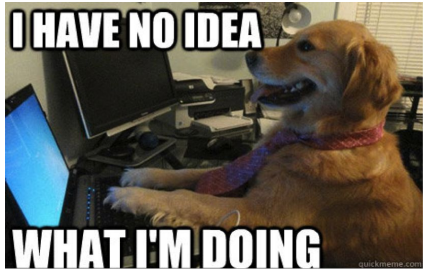
Project 2 Tips

General Tips

- If you’re starting off
  - Go through the Java tutorial from the discussion
  - Helpful to have resultset API from JavaDocs open - find helpful methods
  - Look at discussion slides where we walk through query 0



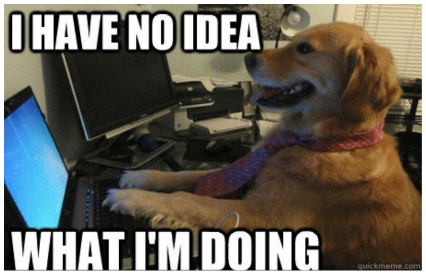
General Tips



- If you're working through the queries
  - Go in order but feel free to skip around
    - Some of the queries have similar logic which you can use to figure them out and reuse code
  - Think about how your queries work in empty/edge cases
  - Pay attention to the way the data is sorted. How can you sort to maximize your ease
  - Make sure you sort in SQL
  - You can check your solution and timings for the public dataset
  - Minimize nested queries and joins
  - Make sure you close all resources

General Tips

- If you’re stuck on a couple queries
  - You may need to use multiple SQL queries in each function
    - Sometimes you’ll need to use nested queries and result set traversals
  - Focus on getting solution right then optimizing as necessary



Using Views

- Some people seem to really like views
  - Easy way to “cache” a query
  - stmt.executeUpdate(“CREATE VIEW view_name AS …”);
    - Different than stmt.executeQuery, no data is returned
  - You can now use your view
  - MAKE SURE YOU DROP YOUR VIEW WHEN YOU ARE DONE WITH IT
    - Otherwise you will have to drop it manually on your sqlplus account every time you test you code on CAEN.
    - Use executeUpdate again for dropping
      - stmt.executeUpdate(“DROP VIEW view_name”);

Queries Timing Out?

- Sometimes you will find that queries you haven’t implemented or previously worked no longer work and time out
  - This is due to the way Oracle and Java play together
  - Make sure you are closing result sets and statements
  - If all else fails add this code to each unimplemented query:

    ```
    ResultSet rst =
    stmt.executeQuery("SELECT * FROM " + UsersTable + " WHERE ROWNUM < 2");
    rst.close();
    stmt.close();
    ```

  - Dummy query that connects to the database, grabs a row, and closes everything

Get started with HW2!

We’re here if you need any help!!

- Office Hours: Schedule is here, both virtual and in person offered
- Piazza
- Next week’s discussion!!!