University of British Columbia, Department of Computer Science

CPSC 304

Cover Page for Project Part 3

Date: October 23, 2018

Group Members:

Name	Student Number	CS Userid	utorial Section	Email Address
Samuel Or	45629300	u4h1b	T1B	or.samuel1@gmail.com
Harry Tao	31267157	k6o0b	T1G	htao1997@hotmail.com
Dante Spadinger-Fengler	14381157	t5z0b	T1C	dantefengler@gmail.com

By typing our names and student numbers in the above table, we certify that the work in the attached assignment was performed solely by those whose names and student IDs are included above.

In addition, we indicate that we are fully aware of the rules and consequences of plagiarism, as set forth by the Department of Computer Science and the University of British Columbia

Project Formal Specification

What Platform Will You Use?

Now that you've had time to think it over more, and perhaps experiment some with PHP & JDBC, it's time to decide which platform that you want to use. In particular, you must say if you are using:

1. The CS UGrad Oracle installation and JDBC
2. The CS UGrad Oracle installation and provided PHP/Apache
3. For other platforms, you must say what you are planning to use. Note that other platforms aren't supported by the Department, so you are responsible for whatever difficulties you will have. (In previous years, this hasn't been a problem for groups, to the best of our knowledge. There have been many successful projects on other platforms.)

   Our platform consists of PostgreSQL and a web application written using NodeJS.

What Functionality Will the Final Application Have?

1. What will the different users be able to do with your system? You don’t have to write the queries in SQL yet, but describe the data and output that each user will see and work with. (More detailed information is in the Deliverables section below.) You’ll want to provide different functionality for different users, even if it’s all driven from the same menu or front-end in your application. Don’t bother creating passwords or separate sign-on screens for

different users; it is OK if you share the same main menu. Here are some examples of what we mean:

- In a banking application, bank employees and customers can see and update different things. We might have one set of SQL statements for the bank employees, and one set for the customers. We don't expect different users to start with different logon IDs—the different users can just select or click on different menu options (like 1 for a withdrawal, 2 for a deposit, 3 to see a monthly statement, etc.)
  - Bank employees might be able to get a statement about their customers and their current balances.
  - Bank customers might be able to get a statement of all their transactions.
- In an airport application, the users might be passengers and airline employees. Employees might assign planes to gates, but passengers wouldn't do this.
- In a medical application, the users might be patients, lab personnel, doctors, etc.
- Even if your application doesn't naturally lend itself to different "classes" of users, then you can still have different features: different windows, different views, and different update activities that junior or less-privileged employees might see.

Deliverables for Your Application

- Deliverable 1: Identify the different kinds of users that your application will service.

  There are two types of users:

- Ordinary registrants on the website. These users can be pet owners, or simply bystanders who search for and report missing pets. These users have the ability to perform most operations except for registering sheltered animals or granting adoption requests.
- Adoption agencies: These users are able to insert and update data for sheltered pets. They also are able to accept or reject adoption applications for pets in this category.

- Itemize the **kinds of queries, reports, or changes to the database** that you expect these users to be able to do. You can add to, or modify, these queries in the future, but at this stage, we want to get a sense of what your application is going to do. You don't need to provide the SQL, yet; but what you have learned about Relational Algebra or SQL so far in the course may help you decide on some of these queries.
    - You need to have **at least 10 SQL data manipulation queries and changes**: some simple, and some complex. You don't have to create the SQL statements yet, but **describe in words what they will be**.

The 10 SQL data manipulation queries and changes that is going to be happening in our database would be ALTER TABLE, SELECT DATA, DELETE TABLE, INSERT MULTIPLE data records and also deleting records in a certain table.

        - More about this requirement and its deliverables are listed below.
    - We expect your application to have the ability to **update** at least some part of the database, *and* the ability to have some queries that depend on the user's input.
        - For example, in a banking domain, I might want the customer to be able to update the database by transferring money from one account to the other, and I might also want a bank employee (e.g., teller) to be able to ask for all of the account information for a given customer. In both cases, the customer number needs to be specified, along with any needed amounts, etc. Don't hard-code these parameters; the user will be entering them.
    - You need to have two or more modification operations that each use a WHERE clause in their respective SQL statements (e.g., DELETE and UPDATE). Describe each of these for your application:
        - Deliverable 2: You need to have at least one INSERT statement.

          A user can insert data of their pet into the database.

          INSERT INTO Pet

          VALUES ("pid", "uid", "name", "M", 2018-10-17, 2018-10-18, "dog", "bull dog", "UjLKKJudcxcvKj1123L")

        - Deliverable 3: You need to have at least one DELETE statement.

          Remove a pet entry from the "Lost" category.

          ```
          DELETE FROM Lost
          WHERE Lost.pid = 3
          ```

- Deliverable 4: You need to have at least one UPDATE statement.

  Update the name and password of a user

  UPDATE User

  SET username ='Tim', password='pass' WHERE pid = 2

  - Queries involving joins: We expect that some of your queries will be something *interesting*, that is, something beyond a simple select, project, or join from a single relation. In other words, you need to have some **meaningful queries that join multiple tables**. Describe them:
- Deliverable 5: At least one meaningful query must join 3 or more tables.

  This is a query that an agency may want to use to find the name of users who adopted a certain breed of pet at a certain campus.

  AdoptionTransaction <- pi_name((sigma_breed = "bull dog" (sigma_postalCode = "a3h7w4" Campus) join_pid Pet) join_pid Adoption) join_uid User)

- Deliverable 6: At least one other meaningful query needs to join 2 or more tables.

  Find the times and locations of all sightings for a specific pet, as well as the pet's name

```
SELECT (Pet.name, Sighting.date, Sighting.lat,
Sighting.long,
FROM ((Pet
INNER JOIN Lost ON Lost.pid = Pet.pid)
INNER JOIN Sighting ON Lost.pid = Sighting.pid)
WHERE Pet.pid = 67;
```

- Deliverable 7: At least one query must be an interesting GROUP BY query (aggregation). Describe it.

  Report the number of pets found on a particular day.

```
SELECT COUNT(pid), foundDate
FROM Found
GROUP BY foundDate;
```

- Deliverables 8-10: Describe the other queries you plan to have (these can be simpler queries), so that you have at least 10 SQL statements overall.

  Find the ids and names of all pets that were reported missing within a certain location.

```
SELECT (Pet.pid, Pet.name)
FROM (Pet
INNER JOIN Lost ON Lost.pid = Pet.pid)
WHERE Lost.lostLat > 49.28 AND Lost.lostLat < 49.48
AND Lost.lostLong > 123.12 AND Lost.lostLong < 123.22;
```

  Generate a list of pets up for adoption at a particular animal shelter

```
SELECT *
FROM Pet, Sheltered
WHERE Pet.pid = Sheltered.pid AND Sheltered.agencyName
= “spca”;
```

  Generate a list of pets that have been lost at a particular date

```
SELECT *
FROM (Pet
INNER JOIN Lost ON Lost.pid = Pet.pid)
WHERE LostDate = 2018-10-17;
```

  Gets data on pets from a certain shelter.

  AdoptablePets <- ((sigma_agencyName = “spca” Campus) join_pid Pet)

  Get data on sightings for a certain pet.

  PetSightings <- ((sigma_pid = “1234” Lost) join_pid Sighting)

  Get data of lost pets around a certain geographic location.

  LostPets <- ((sigma_lostLong > “32” ^ lostLong < “33” ^ lostLat > “32” ^ lostLat < “33”) join_pid Pet)

Get a list of phone numbers of vets with a certain name.

Vets <- pi_phoneNo(sigma_vetName = “Peter” Vet)

SQL SELECT Data statements,SQL INSERT multiple records and SQL ALTER TABLE

- Deliverable 11: You need to have at least one
- view for your database, created using the CREATE VIEW statement in SQL. It should be a proper subset of a table. When the user wants to display all the data from this view, they will get all the columns specified by the view, but not all of the columns in the underlying table.

  This would find all the names of users that are from Vancouver from the User table.

```
CREATE VIEW [Vancouver-User-Names] AS
SELECT Name
FROM User
WHERE City = "Vancouver";
```

  - You don’t have to give the SQL for the view yet; just indicate the subset of which table(s) that will be used, and for which kind of user.
- If you anticipate having any triggers or assertions (read ahead in the textbook), this would also be functionality that would be useful to present as part of your project; but if you don’t have any triggers or assertions or procedures, don’t worry about it because you won’t be deducted marks.

Division of Labour

- Deliverable 12: **What will the division of labour be like among your group?** We want to know how you’re planning on splitting up the work. You should plan for an approximately equal number of tasks, or an equal amount of time required, for each member of your group. Certain members of your group might want to

go over and above this for their own area of responsibility. That's fine; however, don't do another group member's work. For example, you might have these tasks (and possibly others, but you don't have to actually do them now, you just have to say what you plan to do):

Samuel	● Front-end ● Test Queries ● Embed SQL statements and code programming logic
Harry	● Generating data ● Populating the tables ● Embed SQL statements and code programming logic
Dante	● Front-end ● Embed SQL statements and code programming logic

1. Create the tables and other database objects. Be sure to save your SQL scripts, as you are very likely to DROP and re-create your work. In fact, we may ask you to do this, during your demo.
    - Tutorial #6 and its reference material provide some examples of SQL DDL, but you've already seen a bunch of examples in the lectures.
2. Create data for the tables. Then, populate (load) the tables. Be sure to save your scripts. You might want to create a bunch of SQL INSERT statements to load the data. You might even want to write a short program that loops and creates those SQL INSERT statements based on data being read from a file, or that generates the data from scratch (possibly via random number generators).
3. Code each set of queries and test them in SQL (e.g., Oracle's SQL*Plus or another DBMS of your choice).
4. Embed the SQL statements in a program, and code the programming logic. Use a graphical user interface. Format the output appropriately (e.g., in the form of a results window, an HTML page, etc.)

    - The application logic is likely to take up more of the time than any other individual task.
    - All of your group members must take part in this talk because embedded SQL in a host language should be practiced by everyone, and is a learning goal of this course. However, it is OK if some group members do more programming than others, providing those other group members do more of the other tasks.
    - Be prepared to DROP your tables and re-create them, if something significant goes wrong during the UPDATE, INSERT, and DELETE operations.
5. Test each set of queries. Determine how errors will be handled. Take appropriate action. For example, what happens if a user inserts a duplicate key?
6. Document the project.
7. All group members must be present to demo your application to a TA. This will take place near the end of the term.