Rachel Law (861071722)
Ryota Saito (861057726)
Group ID: 24

Project Documentation

Ryota Saito's Contributions

Script fixing: Started project off by fixing the numerous errors found within the script files originally given on ilearn and setting up load_data.sql . Not including the email fixes, many bugs were associated with the way variable types create_tables.sql was set up. For instance, I found a major bug regarding the loading of tables, and the size of the variables being too small. Char(20) was too small for some names in the excel sheet and the cost a whole day trying to make sure the tables were not bugged. They were mostly small inconveniences, but off putting knowing that our program was bugged from the start

CSV creation: the excel document was converted to CSV files with ',' delimiters, since the excel file was incompatible with SQL.

New User Registration/Creation: User creation had to be changed to the user implementing ALL his information on creation instead of inputting just a username, password, and email.

Since almost all of the information in each table was set to NOT NULL, printing out the user profile in any way would always result in errors since the NOT NULL data values were set to NULL. Java threw an exception if I were to replace the null values with a simple string like "none" since I was still referencing a null value in the table. Thus I forced user to input all information at the beginning to avoid future errors when referencing user information.

Adding dates in was also a challenge, as checking if the user actually added a date was a tough task. In the end a regex matches("([0-9]{2})/([0-9]{2})/([0-9]{4})" from stackoverflow helped solve most of the headaches

User Login/Logout: Evaluation of SQL Select return value.

Change Password: Update SQL USR Table.

Search People: Looking up other users was done with searching USR table with query.

Profiles: Lookup all information on all tables and print them out line by line. **Updating Profiles** called updateField() with the necessary information the user wanted to update. This was wrapped with a full menu.

updateField allows the any information in any table beloging to a user to be updated.

IE: updateField(esql, authorisedUser, "company", "WORK_EXPR") updates company in table WORK_EXPR

Message Handling: Sending messages and message creation was easily done with adding queries to the message tables.

Used SELECT * FROM MESSAGE WHERE … And
INSERT INTO MESSAGE(msgId, senderId, receiverId, contents, sendTime, deleteStatus, status

to handle all query executions

Delete Status is handled with :

```
    /*
    Delete Statuses:
    1 = sender delted msg
    2 = reciver deleted
    3 = both sender & reciver deleted
    */
```

Depending on delete status, the message would be invisible to the sender, receiver, or both.

Drafts could not be implemented in this version sadly.

For adding a message query, incrementing a message ID was an issue as Message ID was a primary_key, and thus psql would not allow an increment for the possibility of duplicate primary keys. Instead I got the max value of the col and message ID of a new msg would be max(msgID) + 1 instead.

Menus/inputs: With so many possible inputs being handled in this program, error checking for bad menu inputs took the bulk of my time, rather than the sql logic.

My Cons:
Wished I had started much earlier, knowing that the error checking was going to be the worst part of this project.

My Pros:
Knowing java helped immensely!

Rachel Law

Send Connection Request

- If you have < 5 friends, can send request to any user from the USR Table (except yourself or friends you already have).

- If you have >= 5 friends, can send request to any user within 3 levels of connections to you or friends you already have. We print the available users out for you.
- Updates the CONNECTION_USR Table accordingly
- Grabs lists of Tier1, Tier2, Tier3 (tier being how many direct connections away the user is) for the 3 level requirement
    - No easy SQL query to do this all in one go
- Realized you have to do a lot of queries on the user in order to give the appropriate options and valid user verifications in order to simply send a connection request
    - Query on number of friends currently
    - Query to make sure the user exists under the USR Table
    - Query to make sure not already in their friends list
    - Query based on different connection statuses
        - Also have to UNION between connections from A to B as well as connections from B to A
    - Query to make sure requested user is within 3 connections (for >=5 friends)
- Implemented being able to send connection quests even if the request was previously declined, or is still pending.
    - Had to query first to check if I was going to do an INSERT or UPDATE into the CONNECTION_USR based on if there was already a connection entry between the two users

Accept/Reject Connection Request

- We display any users who have sent you connection requests that are currently still under ‘Request’ status. You can choose to accept or reject them.
- Interesting that opposed to when looking at all pending connections like in ‘send connection request’, only look at connections where some user B has initiated a request to current user A.

View Friends

- Shows you 3 lists (accepted friends, pending friends, rejected friends)
- Not considered a friend until the request is accepted.
- Wanted user to be able to easily distinguish between all types of connections
    - Did a different query for each, but could just put them in one query next time. That would be more efficient.

CSV Files, Tables, Load Data Script

- Cleaning out/Dealing with the mismatching information in the CSV files had me realize the impact of our foreign keys and correspondences between all the tables in our database.
    - Errors in creating/copying over the tables when foreign keys and other restraints are not satisfied.
- So much information to load up. So important to make sure the scripts point to correct paths and work flawlessly while loading tables. Easy to miss small errors

- Encountered that when making the database use varchar(20) to match with char(20) foreign keys, it would takes minutes to build the database. When we switched to completely matching data types char(20) with char(20), database was built in seconds!