CPSC 304 Project Cover Page

Milestone #: 4

Date: Aug 9, 2022

Group Number: 14

Name	Student Number	CS Alias (Userid)	Preferred E-mail Address
Patrick Conner	13781075	w3y6	pconner@student.ubc.ca
James Reinhardt	96645619	i5u3b	james.reinhardt222@gmail.com
Akriti Sharma	79884136	r0s7c	9akriti@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 the case of Project Milestone 0, the main purpose of this page is for you to let us know your e-mail address, and then let us assign you to a TA for your project supervisor.)

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

University of British Columbia, Vancouver
Department of Computer Science

Repository Link:
https://github.students.cs.ubc.ca/CPSC304-2022S-T2/project_i5u3b_r0s7c_w3y6

1. **Please see the first page.**

2. **Please see separate SQL script.**

3. **A PDF file containing:**
    a. **A short description of the final project, and what it accomplished.**

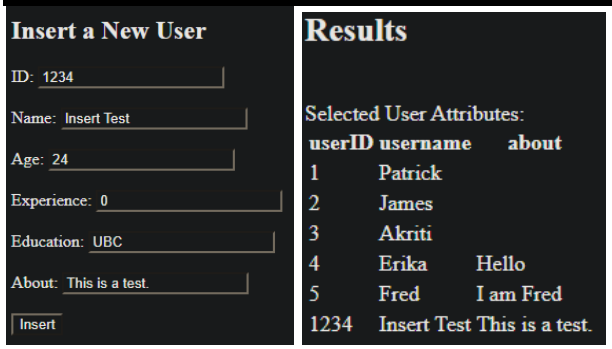
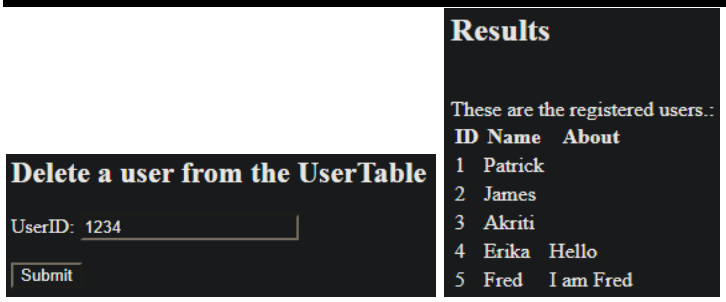
        The final project is a web application built using PHP. Its GUI enables users to populate and query a database containing information on the software development job application process. This includes creating, editing, deleting, and querying Users, as well as querying various Applications, Questions, and Companies.

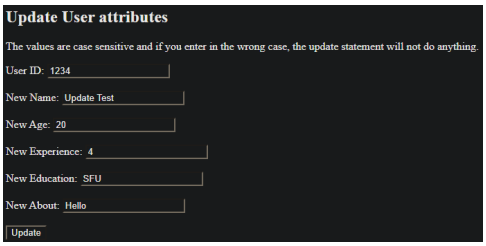
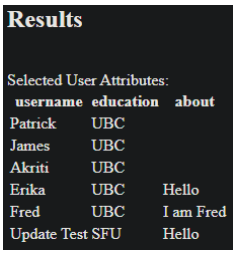
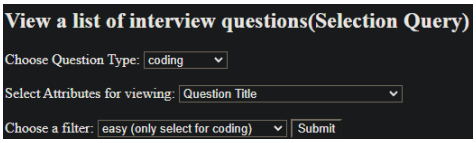
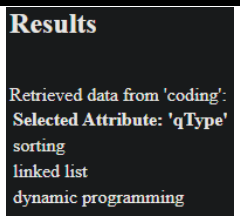
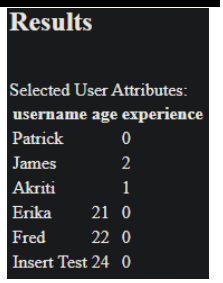
    b. **A description of how your final schema differed from the schema you turned in.**
        i. **If the final schema differed, explain why. Note that turning in a final schema that's different from what you planned is fine, we just want to know what changed and why.**

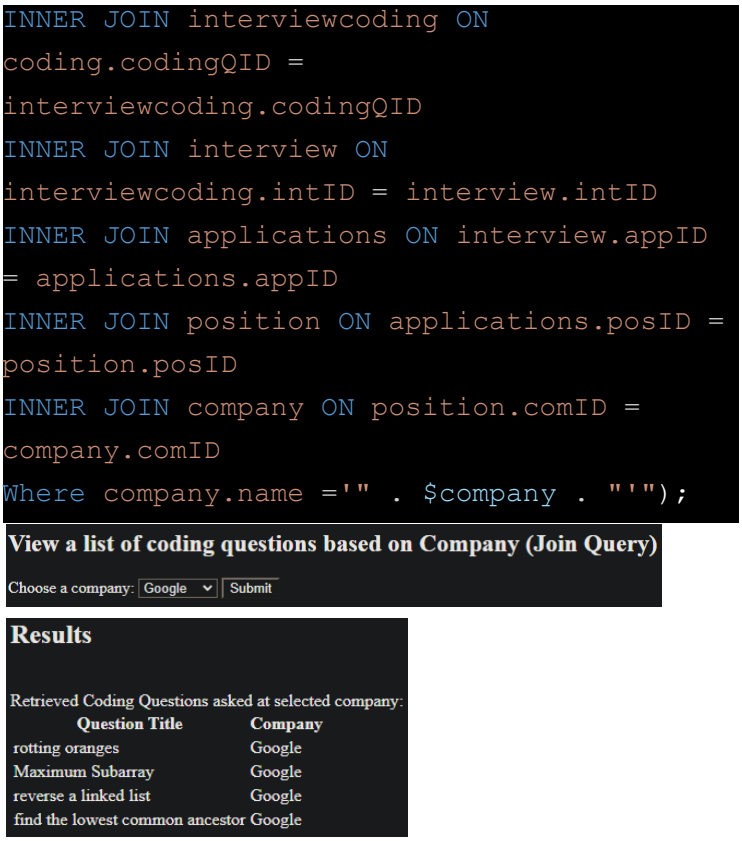
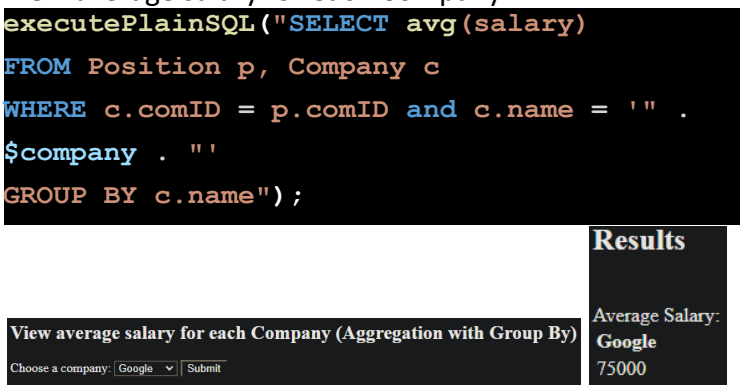
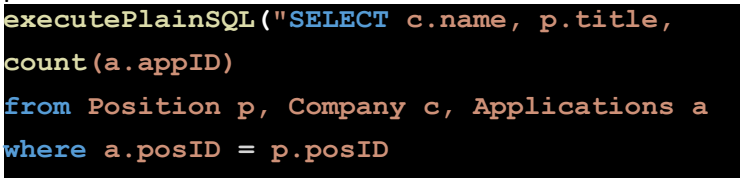
        Only one change was made to the schema, that the Feedback2 table was not implemented. This is because the Feedback2 table only existed for normalization purposes and was unnecessary for the implementation.

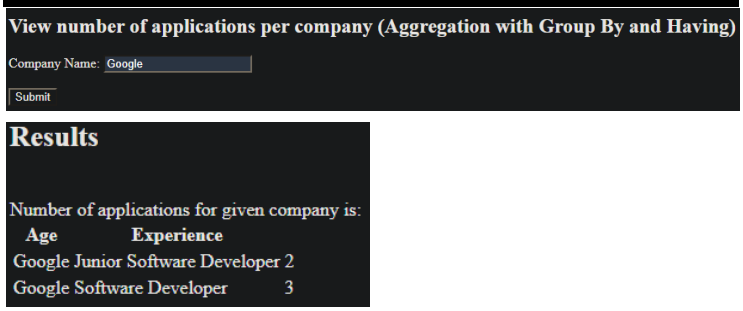
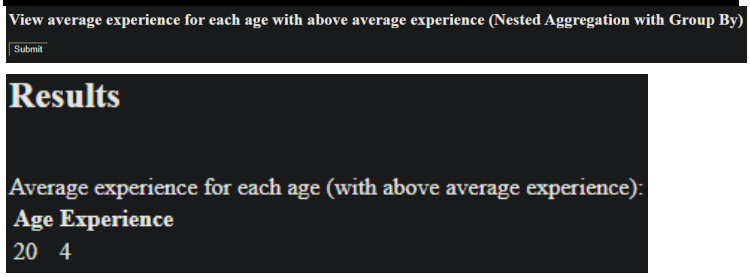
        Additionally, due to time constraints we were not able to fully actualize the GUI as originally intended. Instead, our focus was on actualizing the required SQL queries. Fully realizing the GUI is something that could be accomplished in our personal time, as much of it would be repetition of existing code.

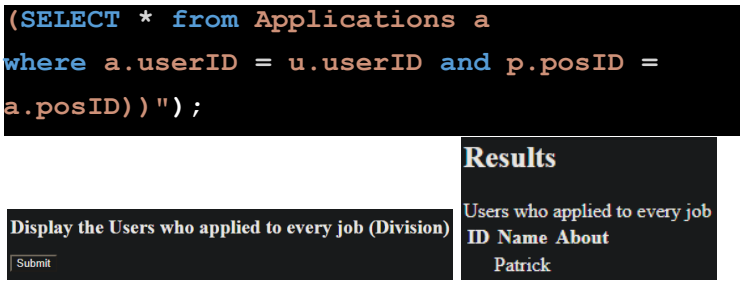
c. List of Queries to Implement:

Query & Requirements:	Description, SQL, and GUI input/output:
INSERT *Provide an interface for the user to specify some input for the insert operation.*	Add a new user - provide an interface to insert all info required to generate a tuple for the User table. `executeBoundSQL("insert into users values (:bind1, :bind2, :bind3, :bind4, :bind5, :bind6)", $alltuples);` 
DELETE *Implement a cascade-on-delete situation (or an alternative that was agreed to by the TA if the DB system doesn't provide this). Provide an interface for the user to specify some input for the deletion operation.*	Provide an interface option to delete a user. `executeBoundSQL("Delete from users where userID = :bind1", $alltuples);` 
UPDATE *Provide an interface for the user to specify some input for the update operation.*	Update User attributes. `executePlainSQL("UPDATE Users SET username='" . $newName . "', age='" . $newAge . "', experience='" . $newExp . "', education='" . $newEdu . "', about='" . $newAbout . "' WHERE userID='" . $userID . "'");`

	 
SELECTION *Create one query of this category and provide an interface for the user to specify the values of the selection conditions to be returned. Example:* *SELECT ...* *FROM ...* *WHERE Field1 = :Var1 AND Field2 > :Var20 = Incorrect or missing*	Select from one of two tables, select attributes to be displayed, and filter to apply. `executePlainSQL("SELECT $attr FROM $tbl WHERE $var='" . $filter . "'");`  
PROJECTION *Create one query of this category, with 3-5 attributes in the projection condition, but not SELECT *. If you wish, you can have the user select the attributes from a drop- down list, but we will accept a hard-coded SELECT statement for the table(s) in question. This can be combined with another step if there are 3-5 attributes in the projection (but not SELECT *).*	Select 3 User attributes to view. `executePlainSQL("SELECT $first, $sec, $third FROM Users");`  
JOIN *Create one query in this category, which joins at least 2 tables and*	Select one of the companies below to view their coding questions: `executePlainSQL("SELECT DISTINCT coding.qTitle,company.name From coding`

performs a meaningful query, and provide an interface for the user to execute this query. The user must provide at least one value to qualify in the WHERE clause (e.g. join the Customer and the Transaction table to find the names and phone numbers of all customers who have purchased a specific item).	
AGGREGATION with GROUP BY *Create one query that requires the use of aggregation (min, max, average, or count are all fine), and provide an interface (e.g., HTML button/dropdown, etc.) for the user to execute this query.*	View average salary for each Company. 
AGGREGATION with HAVING *Create one meaningful query that requires the use of a HAVING clause, and provide an interface (e.g., HTML button/dropdown,*	View number of applications to each company's positions. 

etc.) for the user to execute this query.	`and p.comID = c.comID` `GROUP BY c.name, p.title` `HAVING c.name = '$bind1'");` 
NESTED AGGREGATION with GROUP BY *Create one query that finds some aggregated value for each group (e.g., use a nested subquery, such as finding the average number of items purchased per customer, subject to some constraint). Some examples for the Sailors table are given in the project specs. Note the difference between this query and the above Aggregation Query. You must use separate distinct queries for this criterion and the Aggregation Query (i.e., do not double dip).* *It is fine to use a view to get the desired behaviour.*	View average experience for each age with above average experience. `executePlainSQL("SELECT age,avg(experience)` `FROM Users` `GROUP BY age` `Having avg(experience) >` `(Select avg(experience)` `From Users)");` 
DIVISION *Create one query of this category and provide an interface (i.e., HTML button, etc.) for the user to execute this query (e.g., find all the customers who bought all the items).*	Display users that applied to every position. `executePlainSQL("SELECT u.username` `from Users u` `WHERE NOT EXISTS` `(select p.posID` `from Position p` `WHERE NOT EXISTS`

d. Screenshots of the sample output of the queries using the GUI:
Please see the query table above.

4. Please see separate README.txt file.