University of British Columbia, Vancouver
Department of Computer Science

CPSC 304 Project Cover Page

Milestone #: 3

Date: 03/11/2024

Group Number: 67

Name	Student Number	CS Alias (Userid)	Preferred E-mail Address
Zongxi Li	40628778	zli110	zongxiubc@gmail.com
Shiyu Deng	29343480	b1l9z	simondeng.sy@gmail.comn
Marie Samantha Fidelia	64717309	g0j5n	mariesamantha.f@gmail.com

University of British Columbia, Vancouver
Department of Computer Science

Project Collaboration Web App

Brief Summary

Our project is a project collaboration and management platform with features like task assignment, progress tracking, and communication tools. It utilizes a database to efficiently manage user authentication, project details, and real-time collaboration, to enhance productivity and coordination within teams. The database will provide functionality for storing, retrieving, and managing data related to project collaboration and management.

Timeline

03/15: Group Meeting with TA (Assignees: whole group)

- Fix CS account committing (Assignees: whole group)
- Adjust and finalize timeline and plan according to Jason's feedback
    - Email Jason if any changes are made (Assignee: Zongxi)

03/17: SQL script to create all the tables and data in the database. (Assignee: Simon)

- Need to add constraints that are needed to model some aspects of the ER diagram as well.

03/18:

SQL Queries of INSERT Operation (Assignee: Sam)

- The user should be able to specify what values to insert.
- The insert operation should affect more than one table.
- The chosen query and table(s) should make sense given the context of the application.
- The INSERT operation should be able to handle the case where the foreign key value in the tuple DOES NOT exist in the table that is being referred to. This "handling" can either be that the tuple is rejected by the GUI with an appropriate error message or that the values that are being referred to are inserted.

SQL Queries of DELETE Operation (Assignee: Simon)

- Implement a cascade-on-delete situation.
- The user should be able to choose what values to delete.

03/20:

SQL Queries of UPDATE Operation (Assignee: Simon)

- ○ The user should be able to update any number of non-primary key attributes in a relation.
- ○ The relation used for the update operation must have at least two non-primary key attributes. At least one non-primary key attribute must have either a UNIQUE constraint or be a foreign key that references another table.

SQL Queries of Selection (Assignee: Zongxi)

- ○ The user is able to specify the filtering conditions for a given table. That is, the user is able to determine what shows up in the WHERE clause. It is fine to do these based only on equality, but more complex operations (e.g., less than) are also fine.

SQL Queries of Projection (Assignee: Sam)

- ○ The user is able to choose any number of attributes to view from any relation in the database. Non-selected attributes must not appear in the result.
- ○ One or more tables must contain at least four attributes.

Project Setup (Assignee: Zongxi)

- ○ Set up the Oracle account. (may need help from TAs)

Determine what features will appear in the application (Assignee: whole group)

03/23:

SQL Queries of Join (Assignee: Simon)

- ○ Create one query in this category, which joins at least 2 tables and 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).

UI/UX design based on the features (Assignee: Zongxi)

- ○ Need to include elements like Input Fields, Buttons and display result area, Error Message Field (will probably need to check with TA).

03/25:

SQL Queries of Aggregation with Group By (Assignee: Zongxi)

- ○ 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.

SQL Queries of Aggregation with Having (Assignee: Zongxi)

- ○ Create one meaningful query that requires the use of a HAVING clause, and provide an interface (e.g., HTML button/dropdown, etc.) for the user to execute this query.
- ○ The schema can be statically set but the tuples used in the query cannot be predetermined.

SQL Queries of Nested Aggregation with Group By (Assignee: Sam)

- ○ 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).

SQL Queries of Division (Assignee: Sam)

- ○ 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).

03/27:

User Interface Implementation based on our UI/UX design (Assignee: Zongxi)

- Elements shown in the UI/UX design should be added to the application's main page.

03/29:

Integrate queries into the code.

- Insert (Assignee: Zongxi)
    - The INSERT operation should be able to handle the case where the foreign key value in the tuple does not exist in the table that is being referred to. This "handling" can either be that the tuple is rejected by the GUI with an appropriate error message or that the values that are being referred to are inserted.
- Delete (Assignee: Simon)
    - The chosen query and table(s) should make sense given the context of the application.
- Update (Assignee: Simon)
    - The application should present the tuples that are available so that the user can select which tuple they want to update.
- Selection: (Assignee: Zongxi)
    - The group can choose which table to run this query on. The query and chosen table(s) should make sense given the context of the application.
- Projection (Assignee: Sam)
    - The application must dynamically load the tables from the database (i.e., if we were to insert a new table in the database at the start of the demo, this new table should also show up as a possible option to project from).
- Join: (Assignee: Simon)
    - 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).

04/01

Integrate queries into the code.

- Queries: Aggregation with Group By (Assignee: Zongxi)
    - The schema can be statically set but the tuples used in the query cannot be predetermined.
- Queries: Aggregation with Having (Assignee: Sam)
    - Create one meaningful query that requires the use of a HAVING clause, and provide an interface (e.g., HTML button/dropdown, etc.) for the user to execute this query.

        - The schema can be statically set but the tuples used in the query cannot be predetermined.
    - Queries: Nested Aggregation with Group By (Assignee: Sam)
        - 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 Simon, subject to some constraint).
    - Queries: Division (Assignee: Simon)
        - 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).

04/03

Application checking

- Basic Security Practices: Sanitization (Assignee: Zongxi)
    - Values from the user are not directly used in the database. Basic security practices to prevent injection and rainbow attacks have been followed.
- Basic Error Handling (Assignee: Simon)
    - The user receives notifications about user errors such as trying to insert a duplicate value, invalid input (e.g., invalid characters or an int when only strings are allowed etc.), etc.

04/04

Docs

- Complete README according to MS4 and do final review (Assignee: Sam)
    - Including Cover Page, Repository Link.
    - As well as a Project Description
        - A short description of the final project.
        - A description of how your final schema differed from the schema you turned in. If the final schema differed, explain why.
        - A copy of the schema and screenshots that show what data is present in each relation after the SQL initialization script is run.
- Screenshots of query result (Assignee: Simon)
    - Screenshots of the sample output of the queries using the GUI (for example, for an insertion query you can show what data is in your table

before you run the query, and then show another screenshot after running the query, from some kind of GUI input like a button).

04/05

Review

- ○ Check if the given data is sufficient to demonstrate the functionality of the project. (Assignee: Simon)
- ○ GUI
  - ○ Queries (Assignee: Sam)
    - ○ Queries are designed to only return the data that is needed and in the right order if required. The client does not do any processing of the data such as filtering/sorting etc.
  - ○ User Friendliness (Assignee: Sam)
  - ○ User Notification (Assignee: Simon)
    - ○ The user will receive a success or failure notification upon the completion of an insert, update, delete action and will have a way to verify the action's effect on the database.
- ○ Prepare for TA's questions (assignees: whole group)

Task Breakdown by Categories

- Meeting
  - Meeting with Jason on March 15, Friday.
    - Make changes on our MS3 accordingly.
      - Email TA about the changes we make.
- Project Setup
  - Set up the Oracle account (may need help from TAs).
- Functionality:
  - Determine what features will appear in the application.
- GUI
  - UI/UX design based on the features
    - Need to include elements like Input Fields, Buttons and display result area, Error Message Field.
  - User Interface Implementation based on our UI/UX design
    - Elements shown in the UI/UX design should be added to the application's main page.

- Integrate queries into the code.
    - Insert
        - The INSERT operation should be able to handle the case where the foreign key value in the tuple does not exist in the table that is being referred to. This "handling" can either be that the tuple is rejected by the GUI with an appropriate error message or that the values that are being referred to are inserted.
    - Delete
        - The chosen query and table(s) should make sense given the context of the application.
    - Update
        - The application should present the tuples that are available so that the user can select which tuple they want to update.
    - Selection:
        - The group can choose which table to run this query on. The query and chosen table(s) should make sense given the context of the application.
    - Projection
        - The application must dynamically load the tables from the database (i.e., if we were to insert a new table in the database at the start of the demo, this new table should also show up as a possible option to project from).
    - Join:
        - 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).
    - Queries: Aggregation with Group By
        - The schema can be statically set but the tuples used in the query cannot be predetermined.
    - Queries: Aggregation with Having
        - Create one meaningful query that requires the use of a HAVING clause, and provide an interface (e.g., HTML button/dropdown, etc.) for the user to execute this query.
        - The schema can be statically set but the tuples used in the query cannot be predetermined.
    - Queries: 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).
    - Queries: 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).
    - Basic Security Practices: Sanitization
        - Values from the user are not directly used in the database. Basic security practices to prevent injection and rainbow attacks have been followed.
    - Basic Error Handling
        - The user receives notifications about user errors such as trying to insert a duplicate value, invalid input (e.g., invalid characters or an int when only strings are allowed etc.), etc.
- SQL queries:
    - SQL script to create all the tables and data in the database.
        - Need to add constraints that are needed to model some aspects of the ER diagram as well.
    - Queries of INSERT Operation
        - The user should be able to specify what values to insert.
        - The insert operation should affect more than one table.
        - The chosen query and table(s) should make sense given the context of the application.
        - The INSERT operation should be able to handle the case where the foreign key value in the tuple DOES NOT exist in the table that is being referred to. This "handling" can either be that the tuple is rejected by the GUI with an appropriate error message or that the values that are being referred to are inserted.
    - Queries: DELETE Operation
        - Implement a cascade-on-delete situation.
        - The user should be able to choose what values to delete.
    - Queries: UPDATE Operation
        - The user should be able to update any number of non-primary key attributes in a relation.
        - The relation used for the update operation must have at least two non-primary key attributes. At least one non-primary key attribute must have either a UNIQUE constraint or be a foreign key that references another table.
    - Queries: Selection
        - The user is able to specify the filtering conditions for a given table. That is, the user is able to determine what shows up in the WHERE clause. It is fine to do these based only on equality, but more complex operations (e.g., less than) are also fine.
    - Queries: Projection

        - The user is able to choose any number of attributes to view from any relation in the database. Non-selected attributes must not appear in the result.
        - One or more tables must contain at least four attributes.
    - Queries: Join
        - Create one query in this category, which joins at least 2 tables and 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).
    - Queries: 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.
    - Queries: Aggregation with Having
        - Create one meaningful query that requires the use of a HAVING clause, and provide an interface (e.g., HTML button/dropdown, etc.) for the user to execute this query.
        - The schema can be statically set but the tuples used in the query cannot be predetermined.
    - Queries: 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).
    - Queries: 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).
- Docs
    - Complete README according to MS4 and do final review
        - Including Cover Page, Repository Link, etc.
        - As well as a Project Description:
            - A short description of the final project.
            - A description of how your final schema differed from the schema you turned in. If the final schema differed, explain why.
            - A copy of the schema and screenshots that show what data is present in each relation after the SQL initialization script is run.
    - Screenshots of query result
        - Screenshots of the sample output of the queries using the GUI (for example, for an insertion query you can show what data is in your table before you run the query, and then show another screenshot after running the query, from some kind of GUI input like a button).

- Review:
    - Prepare for TA’s questions.
    - check if the given data sufficient to demonstrate the functionality of the project.
    - Application Implementation
        - Queries
            - Queries are designed to only return the data that is needed and in the right order if required. The client does not do any processing of the data such as filtering/sorting etc.
        - User Friendliness
        - User Notification
            - The user will receive a success or failure notification upon the completion of an insert, update, delete action and will have a way to verify the action's effect on the database.