University of British Columbia, Vancouver
Department of Computer Science

CPSC 304 Project Cover Page

Milestone #: ____4____

Date: ___2024-11-29_________

Group Number: ____15____

Name	Student Number	CS Alias (Userid)	Preferred E-mail Address
Kenny Niu	37151198	y7r6p	kennyn172@gmail.com
Camille Pureza	72136310	p1y7c	camillepureza@gmail.com
Irene Chang	85567402	d0r9y	irecha@student.ubc.ca

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

Milestone 4: Implementation

Brief Project Summary

Our project domain is based on a home management system. The application's features specifically help with a household's item storage, member complaints, tasks, and finances to keep things organized within a shared household. More details on the specific features of our application is under 'What We've Accomplished.'

What We've Accomplished

Backend

We've implemented and tested all required queries (with extra queries for practice) that are useful in our domain. Error messages are returned when errors occurs in the backend. The frontend can access these errors and relay it to the user using the modal component.

Frontend

The front-end successfully does fetch requests to our server and displays any returned values (such as requested tuples from the database). The design is basic and user-friendly. Sections of our project are split into different tabs, which users can click into.

The first tab is to view and remove **household members** registered in the household database. Under the **tasks** tab, users are able to add new tasks and assign them to people. There is also a section to recognize people who are assigned many tasks. We've also added search features for tasks. We have an **Items** section to keep track of the household's item stock. There is also a feature to find all popular items, which can help users keep track which items to buy again on their next grocery store run. In addition, there's a feature to get specific information about items. Under the **financial** tab, we have ways to update certain bills and account details. Furthermore, we have a feature all the **transactions** for a specific person. Lastly, the last tab is a way for the household initiate open discussions about **complaints** to certain people. It lists people who might not be the best roommates!

Final Schema Differences

Store table

- TA suggestion
- Removed storageType and roomType attributes as this information is saved in other tables
- Changed foreign key of room# from ON DELETE SET NULL to ON DELETE CASCADE as room# is part of the primary key and should not be NULL

Consume table

- Removed date from primary key
- Relationship attribute shouldn't be part of the primary key

Items table
- Included dateAcquired attribute to be part of key
- Made necessary changes (tables and re

UtilityBill FD
- Changed periodStartDate, periodEndDate, serviceProviderPlan → cost
    - New FD: periodStartDate, periodEndDate, name → cost
- Using the old FD, we couldn't implement the table in SQL as the serviceProviderPlan would need to be referenced from another table
    - Since serviceProviderPlan isn't a unique attribute, we decided to change it to name instead
    - It also makes sense to change it to name as you have have the same service provider, but have different costs depending on your specific bill
    - For instance, if person A and person B are both under Rogers, they might have different costs depending on what they did during their bill period

Track table
- Changed foreign key of accountID# from ON DELETE SET NULL to ON DELETE CASCADE as accountID# is part of the primary key and should not be NULL

People
- Added NOT NULL to phone number attribute

Constraints & ON UPDATE CASCADE

All our tables with foreign key references should have ON UPDATE CASCADE. However, since Oracle doesn't support this feature, we have not implemented this in our project.

Items has a total participation in Store. We weren't able to simulate the CREATE ASSERTION command using triggers in Oracle. However, we know that if it were implemented, it should make sure that all tuples in Items should be in the Store relationship.

Similarly, Expense has a total participation in Track. Again, if assertion was implemented, then all of Expense tuples should be present in the Track relationship.

Where to Find Required Queries

Insertion
Can be found in appService.js at line 318. There's an extra INSERT query at line 157.
Note: extra query (Insert People) was not implemented in front-end as we have an insertion for DoTask.

Update
Can be found in appService.js at line 172. There's an extra UPDATE query at line 187.
Note: extra query is implemented for fun!

Deletion
Can be found in appService.js at line 273-275.

Selection
Can be found in appService.js at line 365.

Projection
Can be found in appService.js at line 204-206.

Join
Can be found in appService.js at line 291-301.

Aggregation with GROUP BY
Can be found in appService.js at line 377.

Aggregation with HAVING
Can be found in appService.js at line 220-225.

Nested Aggregation with GROUP BY
Can be found in appService.js at line 253-262.

Division
Can be found in appService.js at line 237-243.

Aggregation and Division Queries

Aggregation with GROUP BY
This query returns a table displaying the number of tasks that share a certain attribute (user-specified). For example, the user can choose to see how many tasks are at each location.

Query:
```
SELECT ?, COUNT(*)
        FROM DoTask
        GROUP BY ?
```

? changes based on user dropdown selection. For example, the query could be:
```
SELECT LOCATION, COUNT(*)
        FROM DoTask
        GROUP BY LOCATION
```

Aggregation with HAVING

This query returns roommates who have more than five complaints about them.

Query:

```
SELECT f.toPersonID#, p.name, Count(*)
        FROM FileComplaint f, People p
        WHERE f.toPersonID#=p.ID#
        GROUP BY f.toPersonID#, p.name
        HAVING COUNT(*) > 5
        ORDER BY COUNT(*) DESC, f.toPersonID# DESC
```

Nested Aggregation with GROUP BY

This query returns a leaderboard of the people who have done more tasks than the average person.

```
CREATE OR REPLACE VIEW everyoneNumTasks AS
        SELECT p.id# AS id#, COUNT(*) AS numTasks
        FROM People p, DoTask dt
        WHERE p.id#=dt.personid#
        GROUP BY p.id#

SELECT p.name, e.numTasks
        FROM everyoneNumTasks e, People p
        WHERE p.id#=e.id#
                AND e.numTasks > (
                        SELECT AVG(numTasks)
                        FROM everyoneNumTasks e2
                        )
        ORDER BY e.numTasks DESC
```

Division

This query returns the most popular items within the household; popular items are items that every household member has consumed.

```
SELECT i.name, i.dateAcquired
        FROM Items i
        WHERE NOT EXISTS (
        (SELECT p.ID# FROM People p)
        MINUS
        (SELECT c.personID# FROM Consume c WHERE i.name=c.itemName AND
        i.dateAcquired = c.dateAcquired)
        )
```