University of British Columbia, Vancouver
Department of Computer Science

CPSC 304 Project Cover Page

Milestone #: ____4____

Date: November 29, 2024

Group Number: 18

Name	Student Number	CS Alias (Userid)	Preferred E-mail Address
Matthew Fung	99002214	d0l9u	matthewfung9001@gmail.com
Parsa Seyed Zehtab	84226935	p7m3b	parsaz@shaw.ca
Ethan Wong	14532469	y7n9r	ethanwongca@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/CPSC304-2024W-T1/project_d0l9u_p7m3b_y7n9r

SQL Script for Insert All the Tables and Data In the Database:

Project Description:
Our application focuses on community engagement, local exploration, and tourism. This project will help users discover Vancouver's hidden gems of lesser-known foods, events, and recreation activities encouraging users to explore the whole extent of Vancouver. By providing references like costs, reviews, and transit options, this application will be a city exploration tool that highlights community experiences.

Description of how Final Schema differs:
We added a couple attributes to some objects to improve functionality and design:

1. A title was added to Reviews, to emulate the standards of other review sites.
2. A rating attribute was added to Event, to allow even greater granularity in event queries
3. We removed the FriendsWith relation as by the time we went to implement it, we realized we had all the functionality we needed not only to meet the project requirements but also to create a fantastic user experience.

List of SQL Queries (and where to find them in the code)

2.1.1 INSERT

To meet the criteria for 2.1.1, three functions will be highlighted:
(1) insertWithForeignKeyCheck which can be found in appService.js line 160 … this is the overarching insert function the specific query made in addEvent uses

(2) addEvent which can be found in appService.js line 781

(3) /add-event endpoint, which can be found in appController.js line 589

2.1.2 UPDATE
The query can be found at line 228-233 in appService.js. It allows a user to change a review they wrote.

2.1.3 DELETE
The query can be found at line 264-265 in appService.js. It allows a user to delete a review they wrote.

We also have a delete at line 289 in the same file to allow users to remove notifications to achieve the “on-delete cascade” requirement

2.1.4 SELECTION

To meet the selection query criteria there are three functions / routes to highlight:

1. selectingPlace in the appService.js file which starts on line 303 and ends on line 330 which contains the query
2. ‘/selectingPlace’ route in the appController.js which starts on line 219 and ends on line 295
    a. This function runs sanitization and pre-processes the string for the selectingPlace function
3. Sanitization function used in the ‘/selectingPlace’ route is found in the utils.js file which starts on line 0 and ends on line 30
    a. This is to ensure no dangerous inputs are given from the users

2.1.5 PROJECTION

To meet the projection query criteria there are three functions to highlight:

1. projectFromPlace in the appService.js which starts on line 332 and ends on line 366 which contains the query
2. ‘/projectFromPlace’ route in the appController.js which starts on line 297 and ends on line 320.

2.1.6 Join

To meet join, the functions to highlight are:

1. getEventsAtPlace in appService.js line 378, which contains the query
2. /place-events endpoint found on line 330 in appController.js

Queries 2.1.7 - 2.1.8 with descriptions

2.1.7 Aggregation with GROUP BY

The query we used can be found in the function getAverageEventRatingPerPlace in appService.js line 426 for the function, with the query being:

```
SELECT e.Name, e.Address, AVG(e.Rating) AS average_rating
```

```
FROM Event e
GROUP BY e.Name, e.Address
HAVING AVG(e.Rating) IS NOT NULL
```

This query retrieves the average rating of events of each place. It excludes events with no ratings using the HAVING clause, and ultimately allows users to see the average rating of events at each place.

2.1.8 Aggregation with HAVING

To meet the Aggregation with Having quota here are the following functions:

1. getCuisineAboveThreshold in appService.js on line 441 and ends on line 477 and contains the query
2. /getCuisineAboveThreshold in appController.js on line 366 and ends on line 388 and contains the endpoint

The query in itself can be found on line 455 in the appService file

```
SELECT r.Cuisine, AVG(rv.Rating) AS AverageRating
FROM Restaurant r
JOIN Reviews rv ON r.Name = rv.Name AND r.Address = rv.Address
GROUP BY r.Cuisine
HAVING AVG(rv.Rating) >= :threshold
```

This query groups each restaurant by cuisines using the GROUP BY clause, in Vancouver and allows the user to find a cuisine above a specific threshold, using the HAVING CLAUSE of what the user wants.

2.1.9 Nested Aggregation with GROUP BY

This query can be found at line 466 in appService.js. This query is used to determine the restaurant that has the highest average review. If there is a tie, they are all shown.

```
SELECT p.Name, p.Address, AVG(r.Rating) AS AvgRating
FROM Place p
JOIN Restaurant res ON p.Name = res.Name AND p.Address = res.Address
JOIN Reviews r ON p.Name = r.Name AND p.Address = r.Address
GROUP BY p.Name, p.Address
HAVING AVG(r.Rating) = (
                    SELECT MAX(AvgRating)
                    FROM (
                        SELECT AVG(r.Rating) AS AvgRating
                        FROM Place p
```

```
                        JOIN Restaurant res ON p.Name = res.Name AND
p.Address = res.Address
                        JOIN Reviews r ON p.Name = r.Name AND p.Address =
r.Address
                        GROUP BY p.Name, p.Address
                    )
                )
ORDER BY p.Name
```

2.1.10 Division

To support division, the function that calls the function getPlacesReviewedByAll which has the query, and can be found in appService.js line 536. The query is:

```
        SELECT P.Name, P.Address
        FROM Place P
        WHERE NOT EXISTS (
            SELECT U.UserID
            FROM Users U
            WHERE NOT EXISTS (
                SELECT R.Name, R.Address
                FROM Reviews R
                WHERE R.Name = P.Name
                AND R.Address = P.Address
                AND R.UserID = U.UserID
            )
        )
    `;
```

This query retrieves all places (Name, Address) that have been reviewed by **every** user in the Users table. It uses nested NOT EXISTS conditions to implement the division logic, ensuring that for every user, a corresponding review exists for each place.