University of British Columbia, Vancouver
Department of Computer Science

CPSC 304 Project - Transit System

Milestone #: 4

Date: November 29th, 2024

Group Number: 49

Name	Student Number	CS Alias (Userid)	Preferred E-mail Address
Dayshaun Lee	76084086	o4h1m	lee.dayshaun@gmail.com
Minh Vu	33077769	p5n0o	minhvuams@gmail.com
Khue Do	70790423	k3h7x	khuedothi@gmail.com

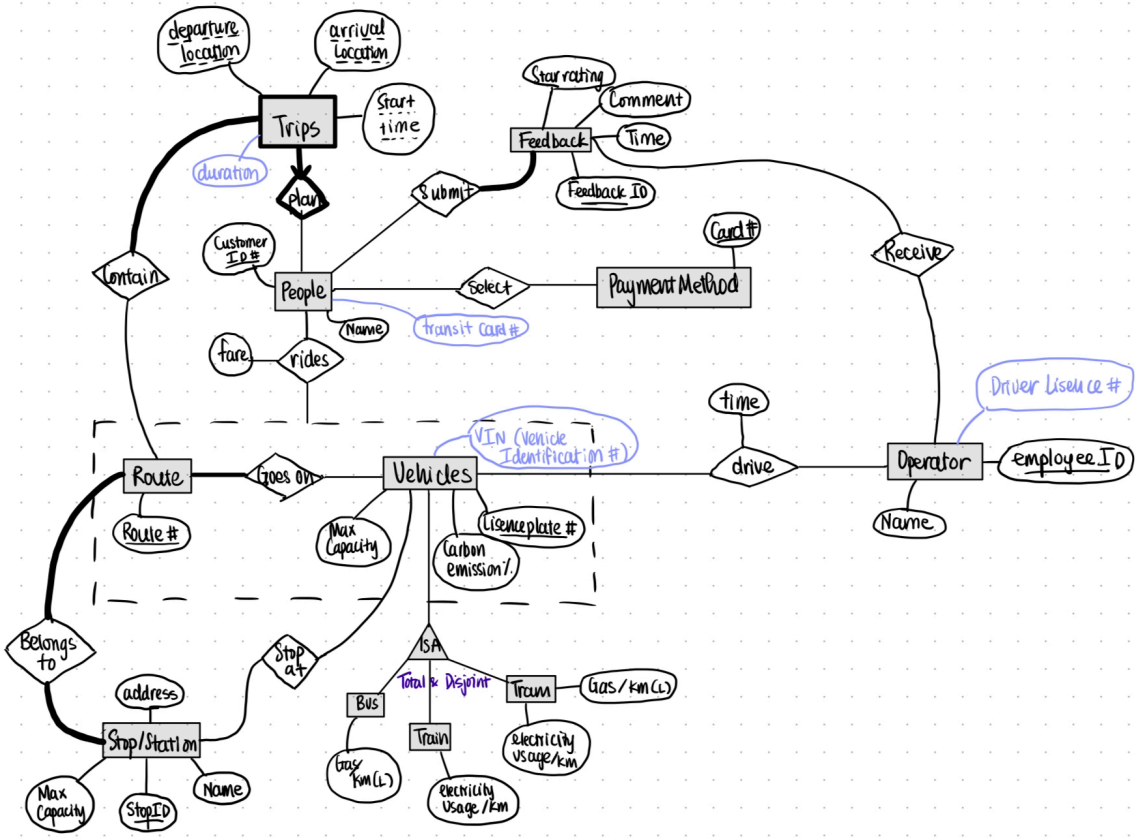
By typing our names and student numbers in the above table, we certify that the work in the attached assignment was performed solely by those whose names and student IDs are included above. (In the case of Project Milestone 0, the main purpose of this page is for you to let us know your e-mail address, 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

I. Project Summary

Repo link: https://github.students.cs.ubc.ca/CPSC304-2024W-T1/project_k3h7x_o4h1m_p5n0o

This project aims to optimize the Vancouver public transportation system by providing real-time scheduling data, operational vehicles and routes information, user feedback and environmental impact tracking through an Oracle database. It will manage bus and train schedules, vehicle maintenance, and ridership data to improve efficiency and wait times for customers, at the same time meet the city's sustainability goals.



University of British Columbia, Vancouver
Department of Computer Science

List of what needs to be included in this PDF (from project description doc)

- A short description of the final project, and what it accomplished.
- A description of how your final schema differed from the schema you turned in. If the final schema differed, explain why.
- A list of all SQL queries used to satisfy the rubric items and where each query can be found in the code (file name and line number(s)).
- For SQL queries 2.1.7 through 2.1.10 inclusive, include a copy of your SQL query and a maximum of 1-2 sentences describing what that query does. You can embed this in your above list of queries. You don't need to include the output of the query

II. Any updates to original Schemas

Some table Names or Attribute Names have been modified to make the code work/consistency. However they are not that much different and definitely interpretable from original schemas. For example "gas/km" has been changed to "gas_km". the "comment" attribute for Feedback was changed to "feedbackComment" since "comment" is a reserved keyword in SQL.
Some tables also had incorrect foreign keys, so those tables have been fixed (for example, TripsPlan1 and TripsPlan2 now have FKs that reference each other and the People relation).

Other structural characteristics stay the same

III. List of all SQL Queries

All queries can be found in the appService.js file

- 2.1.1: INSERT (line 235)

```
INSERT INTO ${tableName} (${columnsList}) VALUES (${placeholders})
```

Note: we made this query dynamic so it can insert anything from any table. The query used in the project is

```
INSERT INTO SelectPayment(cardNumber, customerID) VALUES (${placeholders})
```

- 2.1.2: UPDATE (line 285)

```
UPDATE Vehicles SET ${setClauses} WHERE licensePlateNumber =
:licensePlateNumber
```

- 2.1.3: DELETE (line 260)

```
DELETE FROM Operator WHERE employeeID = :employeeID
```

- 2.1.4: SELECTION (line 134)

```
SELECT DISTINCT ${selectedAttributes} FROM Stops WHERE ${condition}
```

- 2.1.5: PROJECTION (line 164)

```
SELECT ${attributes} FROM Feedback
```

- 2.1.6: JOIN (178-180)

```
SELECT DISTINCT tp.startTime, tp.arrivalLocation, tp.departureLocation
        FROM TripsPlan2 tp, People p
        WHERE p.customerID=tp.customerID AND p.peopleName=${name} AND
        p.transitCardNumber=${transitCardNumber}`
```

- 2.1.7: Aggregation with GROUP BY

  appService.js (348-354)

  For each arrivalLocation, find the shortest duration Trip with a duration >= minDuration, and the associated departureLocation for that Trip.

```
SELECT arrivalLocation, departureLocation, duration
            FROM TripsPlan1
            WHERE (arrivalLocation, duration) IN (
                SELECT arrivalLocation, MIN(duration)
                FROM TripsPlan1
                WHERE duration >= :minDuration
                GROUP BY arrivalLocation
            )
```

- 2.1.8: Aggregation with HAVING
  appService.js(373-377)
  For each operator, find the average rating, with a rating of at least minRating

```
SELECT r.employeeID, AVG(f.starRating) as avgStarRating
            FROM Feedback f, Receiver
            WHERE f.feedbackID=r.feedbackID
            GROUP BY r.employeeID
            HAVING AVG(f.starRating)>=:minRating
```

- 2.1.9: Nested aggregation with GROUP BY
  appService.js(394-404)
  Vehicles travel on routes and release carbon emissions. Find the routeNumber(s) that have the highest average carbon emissions for Vehicles that travel on it.

```
 SELECT g.routeNumber, AVG(v.carbonEmission) AS avgCarbonEmission
            FROM Vehicles v, GoesOn g
            WHERE g.licensePlateNumber=v.licensePlateNumber
            GROUP BY g.routeNumber
            HAVING AVG(v.carbonEmission)
                >=all(
                    SELECT AVG(v2.carbonEmission)
                    FROM Vehicles v2, GoesOn g2
                    WHERE g2.licensePlateNumber=v2.licensePlateNumber
                    GROUP BY g2.routeNumber)
```

- 2.1.10: DIVISION
  appService.js(308-320)
  Find the employeeID and Name of the Operator that received the most diverse ratings. This means Operators that have received AT LEAST 5 feedbacks, which corresponds to all the available starRatings (1,2,3,4,5).

```
SELECT DISTINCT O.employeeID, O.operatorName
                FROM Operator O
                JOIN Receive R ON O.employeeID = R.employeeID
                JOIN Feedback F ON R.feedbackID = F.feedbackID
                WHERE NOT EXISTS (
                    SELECT SR.starRating
                    FROM (SELECT DISTINCT starRating FROM Feedback) SR
                    WHERE NOT EXISTS (
                        SELECT 1
                        FROM Receive R2
                        JOIN Feedback F2 ON R2.feedbackID = F2.feedbackID
                        WHERE R2.employeeID = O.employeeID
                        AND F2.starRating = SR.starRating
                    )
                )
```