Cababa: A Cab Booking System

Akanksha Singal (2021008), Ananya Goyal (2021011)

1.1 Embedded SQL Queries

Several embedded SQL queries have been added that can be executed using the front end. TWO of them are:

1. Used for authentication of the user while logging in

```
SELECT * FROM users WHERE user_email = %s AND pswd = %s
```

2. Used to create an account on the signup page and insert information in the database

```
INSERT INTO users (user_email, first_name, last_name, phno, pswd)
VALUES (%s, %s, %s, %s, %s)
```

1.2 OLAP Queries

The following OLAP queries have been added to the platform:

1. This is an OLAP (Online Analytical Processing) query in MySQL that calculates the total distance covered and the average rating of drivers by week. The query uses the YEARWEEK() function to group the rides by week and the SUM() function to calculate the total distance each driver covers during that week. It also uses the AVG() function to calculate the average rating of each driver during that week.
   The query aggregates and summarises data to provide insights into drivers' weekly performance. It groups the rides by multiple dimensions (driver, week, and distance) and calculates the total distance covered and the average rating for each combination of driver and week.
   The GROUP BY clause specifies the columns that the data is grouped by (final_rating, first_name, last_name, driver_email, week, and distance), while the ORDER BY clause specifies the order in which the data is sorted (in descending order of week).

```
SELECT d_email, YEARWEEK(pickup_time) as week, SUM(distance),
AVG(final_rating) as rating
FROM rides
JOIN drivers ON rides.d_email = drivers.driver_email
GROUP BY drivers.final_rating, drivers.first_name, drivers.last_name,
drivers.driver_email, week, distance
ORDER BY week DESC;
```

2. This query shows the total earnings per day for the last four months. The query selects the ride date and the total earnings made on that date by joining the rides, vehicles, and vehicle_types tables on their corresponding foreign keys. The WHERE clause is filtering the data to include only the rides that happened in the last 4 months.
   Then the query performs the aggregation by grouping the results by the ride date using the GROUP BY clause. Finally, the ORDER BY clause sorts the results in descending order by the ride date.

```
SELECT DATE(r.pickup_time) AS ride_date, SUM(fare) AS total_earnings
       FROM rides r
        JOIN vehicles v ON r.vehicle_no = v.reg_no
        JOIN vehicle_types vt ON v.type_id = vt.type_id
        WHERE DATE(r.pickup_time) >= DATE_SUB(NOW(), INTERVAL 4
MONTH)
        GROUP BY ride_date
        ORDER BY ride_date DESC;
```

3. Show groups of the rides by the year and pickup location, and calculate the total fare for each group. This OLAP query uses the ROLLUP operator to perform aggregation on multiple dimensions of the data.
   The query selects the year of the pickup time, pickup location, and the total fare for each year and location. The ROLLUP operator then generates subtotals and grand totals for the data, producing a report summarising the data by year and location. The GROUPING() function indicates whether a row is a subtotal or the grand total. When the GROUPING() function returns a value of 1 for a column, it indicates that the row is a subtotal or grand total for that column.
   This OLAP query provides insights into the revenue generated by each location over the years and the revenue trend for each location. The output shows the subtotal for each year and the grand total for all the data, enabling business users to make data-driven decisions on allocating resources to the most profitable locations.

```
SELECT YEAR(pickup_time), pickup_loc, SUM(fare) as fare,
    GROUPING(YEAR(pickup_time)) AS grp_year,
    GROUPING(pickup_loc) AS grp_pickup_loc
    FROM rides
    GROUP BY YEAR(pickup_time), pickup_loc WITH ROLLUP;
```

4. This query shows the top 3 drivers by total ride fare and year. This is an OLAP query because it involves aggregating data from multiple tables and summarising the results to provide insight into business operations. The query retrieves data about drivers, rides, fares, and ratings and groups it by driver, year, and rating. It then orders the results by total fare and selects the top three drivers.

   The query is useful for analysing which drivers generated the most revenue in a given year and how their final ratings may have influenced their earnings. Using the LIMIT clause limits the results to the top three drivers, making it easier to identify the most successful drivers.

```
        SELECT drivers.driver_email, drivers.first_name,
drivers.last_name, YEAR(pickup_time) as year, SUM(fare) as
total_fare, final_rating as rating
        FROM rides
        JOIN drivers ON rides.d_email = drivers.driver_email
        GROUP BY drivers.driver_email, drivers.first_name,
drivers.last_name, year, final_rating
        ORDER BY total_fare DESC
        LIMIT 3;
```

5. Cube query to analyse cab bookings by date, distance, and pickup location. This is an OLAP query that uses the CUBE operator to perform a multi-dimensional analysis of cab bookings data by date, distance, and pickup location. The CUBE operator generates a result set that includes all possible combinations of the specified grouping columns, thereby enabling the analysis of data at different levels of granularity.
   The SELECT statement retrieves the year of the pickup time, distance, pickup location, and the count and sum of rides for each combination of these dimensions. The GROUP BY clause uses the CUBE operator to group the data by all possible combinations of the year of the pickup time, distance, and pickup location dimensions.

   The resulting output will include subtotals and grand totals for each level of granularity, which can be useful in identifying patterns and trends in the data. For example, the output can determine which pickup locations have the highest ride counts and total fares for different distances and years or which years have the highest ride counts and total fares for different distances and pickup locations.

```
SELECT YEAR(pickup_time), distance, pickup_loc, COUNT(*) AS
ride_count, SUM(fare) AS total_fare
FROM rides
GROUP BY CUBE(YEAR(pickup_time), distance, pickup_loc);
SELECT YEAR(pickup_time), distance, pickup_loc, COUNT(*) AS
ride_count, SUM(fare) AS total_fare
```

```
FROM rides
GROUP BY CUBE(YEAR(pickup_time), distance, pickup_loc);
```

1.3 Triggers

1. ratings_update trigger is activated when the ratings of the ride are being updated in the rides table. Before the update of the ratings in rides, the net ratings, total rated trips, final ratings and total trips are updated in the drivers table.

```
DELIMITER #
CREATE TRIGGER ratings_update
BEFORE UPDATE ON rides
FOR EACH ROW
BEGIN
IF NEW.rating > 0 THEN
UPDATE drivers
SET rated_trips = rated_trips + 1, net_rating_sum = net_rating_sum +
NEW.rating,
final_rating = net_rating_sum/rated_trips, total_trips = total_trips
+ 1
WHERE driver_email = NEW.d_email;
END IF;
END#
DELIMITER ;
```

2. settings_update trigger is activated when a row in the users table is updated. Before updating the attributes of a particular record, all the rides taken by the user, as referred to by the user_email before the update, are deleted from the rides table. This means that when a user updates their email address, phone number and password, all the previous ride history linked to the old email address will no longer be visible and be deleted from the database.

```
DELIMITER #
CREATE TRIGGER settings_update
BEFORE UPDATE ON users
FOR EACH ROW
BEGIN
DELETE FROM rides
WHERE u_email = OLD.user_email;
END#
DELIMITER ;
```