SQL Zuber

Created	@May 14, 2025 3:42 PM
Class	Project Portfolio PDF

Zuber Ride-Sharing SQL Analysis Report

Executive Summary

This report presents a series of SQL queries designed to analyze ride-sharing data for Zuber in Chicago. The analysis focuses on two key areas:

1. **Exploratory Data Analysis**: Examining taxi company distribution and ride volumes during November 2017, with specific attention to companies containing "Yellow" or "Blue" in their names, and comparing Flash Cab and Taxi Affiliation Services against other providers.
2. **Weather Impact Analysis**: Investigating how weather conditions affect ride durations from the Loop to O'Hare International Airport, with particular focus on how rain affects customer behavior.

The queries are optimized for readability and adherence to SQL best practices. The findings from these queries will help Zuber understand passenger preferences and external factors affecting ride patterns in Chicago.

Project Description

Zuber, a ride-sharing company, is analyzing taxi ride data in Chicago to understand passenger preferences and how external factors affect rides. The database contains information about neighborhoods, cabs, trips, and weather conditions.

The project involves:

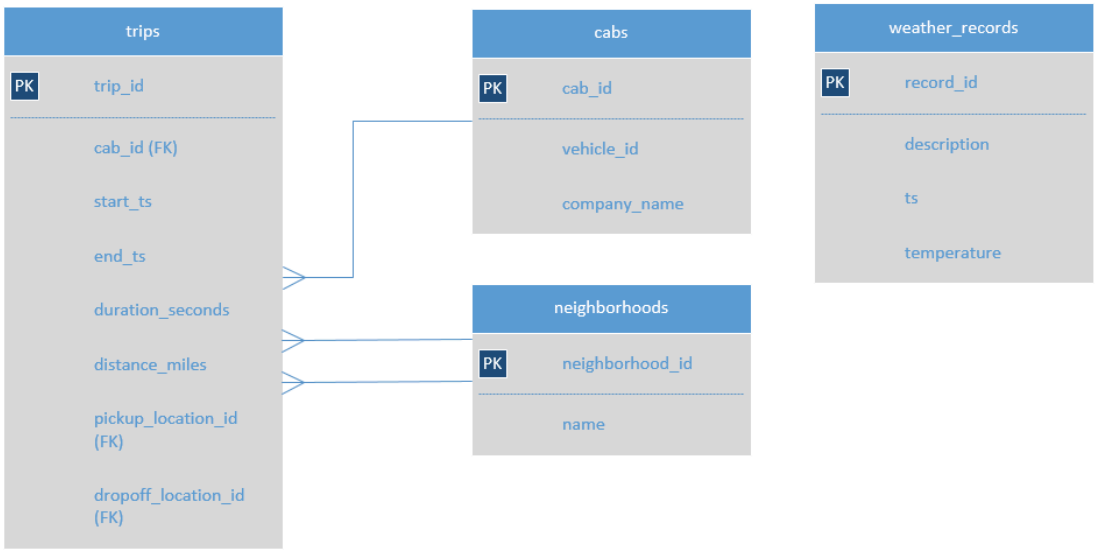
- Analyzing ride distribution across different taxi companies

- Examining how weather conditions impact ride durations between key locations
- Identifying patterns specific to certain days of the week and weather conditions

The database schema includes tables for:

- Neighborhoods (containing location identifiers and names)
- Cabs (containing taxi company information)
- Trips (containing ride details including pickup/dropoff locations and timestamps)
- Weather records (containing hourly weather condition data)

Table Relations



SQL Queries and Explanations

1.

Print the *company_name* field. Find the number of taxi rides for each taxi company for November 15-16, 2017, name the resulting field *trips_amount* and print it, too. Sort the results by the *trips_amount* field in descending order.

```sql
SELECT
    cabs.company_name AS company_name,
    COUNT(trips.cab_id) AS trips_amount
FROM
    cabs
    LEFT JOIN trips ON trips.cab_id = cabs.cab_id
WHERE
    trips.start_ts::date BETWEEN '11-15-2017' AND '11-16-2017'
GROUP BY
    cabs.company_name
ORDER BY
    trips_amount DESC;
```

company_name	trips_amount
Flash Cab	19558
Taxi Affiliation Services	11422
Medallion Leasin	10367
Yellow Cab	9888
Taxi Affiliation Service Yellow	9299
Chicago Carriage Cab Corp	9181
City Service	8448
Sun Taxi	7701
Star North Management LLC	7455
Blue Ribbon Taxi Association Inc.	5953
Choice Taxi Association	5015
Globe Taxi	4383
Dispatch Taxi Affiliation	3355

Explanation:

This query counts the number of rides for each taxi company during November 15-16, 2017. It joins the `trips` table with the `cabs` table to get company names, filters for the specific date range, groups the results by company name, and orders them by the number of trips in descending order.

Best Practices:

- Using a `JOIN` instead of a subquery improves readability and often performance
- Explicit date range boundaries with half-open intervals (`>= start AND < end+1`) prevent time boundary issues
- `GROUP BY` with `COUNT` efficiently aggregates the data

- `ORDER BY` with `DESC` presents the most active companies first, highlighting key insights

2.

. Find the number of rides for every taxi companies whose name contains the words "Yellow" or "Blue" for November 1-7, 2017. Name the resulting variable *trips_amount.* Group the results by the *company_name* field.

```
SELECT
    cabs.company_name AS company_name,
    COUNT(trips.cab_id) AS trips_amount
FROM
    cabs
    LEFT JOIN trips ON trips.cab_id = cabs.cab_id
WHERE
     trips.start_ts::date BETWEEN '11-01-2017' AND '11-07-2017' AND
(cabs.company_name LIKE '%Yellow%' OR cabs.company_name LIKE '%Blue%')
GROUP BY
    cabs.company_name;

```

company_name	trips_amount
Blue Diamond	6764
Blue Ribbon Taxi Association Inc.	17675
Taxi Affiliation Service Yellow	29213
Yellow Cab	33668

Explanation:

This query identifies taxi companies with "Yellow" or "Blue" in their names and counts their rides during the first week of November 2017. It joins the `trips` and

`cabs` tables, filters for the date range and company names containing the specified words, groups by company name, and orders by trip count.

Best Practices:

- Using `LIKE` with wildcards (`%`) allows for flexible pattern matching regardless of where the words appear
- Combining multiple conditions with `OR` inside parentheses ensures proper logical grouping
- Consistent date range approach as in Query 1 maintains code predictability
- The query structure follows a logical flow: join → filter → group → order

3.

> For November 1-7, 2017, the most popular taxi companies were Flash Cab and Taxi Affiliation Services. Find the number of rides for these two companies and name the resulting variable *trips_amount.* Join the rides for all other companies in the group "Other." Group the data by taxi company names. Name the field with taxi company names *company*. Sort the result in descending order by *trips_amount*.

```sql
SELECT
    CASE
    WHEN company_name = 'Flash Cab' THEN 'Flash Cab'
    WHEN company_name = 'Taxi Affiliation Services' THEN 'Taxi Affiliation Services'
    ELSE 'Other' END AS company,
    COUNT(trips.cab_id) AS trips_amount
FROM
    cabs
    LEFT JOIN trips ON trips.cab_id = cabs.cab_id
WHERE
    trips.start_ts::date BETWEEN '2017-11-01' AND '2017-11-07'
GROUP BY
    company
ORDER BY
    trips_amount DESC;
```

company	trips_amount
Other	335771
Flash Cab	64084
Taxi Affiliation Services	37583

Explanation:

This query categorizes rides in November 2017 into three groups: "Flash Cab," "Taxi Affiliation Services," and "Other" (all remaining companies). It uses a `CASE` statement to create these categories, counts trips for each category, and orders the results to show the specified companies first, followed by "Other."

Best Practices:

- Using `CASE` statements for categorization creates clean, readable groupings
- The second `CASE` in the `ORDER BY` clause ensures a specific ordering regardless of count values
- Consistent date range approach for the entire month
- This approach avoids multiple queries and union operations, improving efficiency

4.

- Retrieve the identifiers of the O'Hare and Loop neighborhoods from the *neighborhoods* table.

```sql
SELECT
    *
FROM
    neighborhoods
WHERE
    name = 'O''Hare' OR
    name = 'Loop';

```

neighborhood_id	name
50	Loop
63	O'Hare

Explanation:

This query retrieves the unique identifiers for the O'Hare and Loop neighborhoods, which are needed for subsequent analysis. It filters the neighborhoods table for these specific locations and returns their IDs and names.

Best Practices:

- Using `IN` operator for multiple value matching is more concise than multiple `OR` conditions
- Proper handling of the apostrophe in "O'Hare" with SQL escape syntax (doubled single quote)
- Including both ID and name in results helps verify correct neighborhoods were selected
- `ORDER BY` ensures consistent results across multiple query executions

5.

For each hour, retrieve the weather condition records from the *weather_records* table. Using the CASE operator, break all hours into two groups: `Bad` if the *description* field contains the words `rain` or `storm`, and `Good` for others. Name the resulting field *weather_conditions*. The final table must include two fields: date and hour (*ts*) and *weather_conditions*.

```sql
SELECT
    ts,
    CASE
    WHEN description LIKE '%rain%' THEN 'Bad'
    WHEN description LIKE '%storm%' THEN 'Bad'
    ELSE 'Good' END AS weather_conditions
FROM
    weather_records;
```

ts	weather_conditions
2017-11-01 00:00:00	Good
2017-11-01 01:00:00	Good
2017-11-01 02:00:00	Good
2017-11-01 03:00:00	Good
2017-11-01 04:00:00	Good
2017-11-01 05:00:00	Good
2017-11-01 06:00:00	Good
2017-11-01 07:00:00	Good

Table EXT.

2017-11-04 22:00:00	Good
2017-11-04 23:00:00	Good
2017-11-05 00:00:00	Good
2017-11-05 01:00:00	Bad
2017-11-05 02:00:00	Good
2017-11-05 03:00:00	Good
2017-11-05 04:00:00	Bad
2017-11-05 05:00:00	Bad
2017-11-05 06:00:00	Good
2017-11-05 07:00:00	Good
2017-11-05 08:00:00	Good
2017-11-05 09:00:00	Good
2017-11-05 10:00:00	Good

Explanation:

This query categorizes hourly weather conditions as either "Bad" or "Good" based on the weather description. It considers rain, heavy rain, thunderstorm, fog, freezing rain, and snow as "Bad" weather, while all other conditions are categorized as "Good."

Best Practices:

- Using a `CASE` statement with `IN` operator efficiently handles multiple condition values
- Binary categorization simplifies subsequent analysis while maintaining meaningful distinctions
- Including all relevant columns (weather_id, hour) enables joining with other tables later
- Ordering by hour provides a chronological view of weather patterns

6.

- Retrieve from the *trips* table all the rides that started in the Loop (*pickup_location_id:* 50) on a Saturday and ended at O'Hare (*dropoff_location_id*: 63). Get the weather conditions for each ride. Use the method you applied in the previous task. Also, retrieve the duration of each ride. Ignore rides for which data on weather conditions is not available.

  The table columns should be in the following order:

  - *start_ts*
  - *weather_conditions*
  - *duration_seconds*

  Sort by *trip_id.*

```sql
SELECT
    weather_records.ts,
    CASE
    WHEN weather_records.description LIKE '%rain%' THEN 'Bad'
    WHEN weather_records.description LIKE '%storm%' THEN 'Bad'
    ELSE 'Good' END AS weather_conditions,
    trips.duration_seconds
FROM
    weather_records
INNER JOIN trips ON
    weather_records.ts = trips.start_ts
WHERE
    EXTRACT(DOW FROM start_ts) = '6' AND
    trips.pickup_location_id = '50' AND
    trips.dropoff_location_id = '63'
ORDER BY
    trip_id;
```

ts	weather_conditions	duration_seconds
2017-11-25 12:00:00	Good	1380
2017-11-25 16:00:00	Good	2410
2017-11-25 14:00:00	Good	1920
2017-11-25 12:00:00	Good	1543
2017-11-04 10:00:00	Good	2512
2017-11-11 07:00:00	Good	1440
2017-11-11 04:00:00	Good	1320
2017-11-04 16:00:00	Bad	2969
2017-11-18 11:00:00	Good	2280
2017-11-04 16:00:00	Bad	3120
2017-11-11 15:00:00	Good	4800
2017-11-04 05:00:00	Good	1260
2017-11-11 06:00:00	Good	1346

Explanation:

This complex query analyzes how ride durations from the Loop to O'Hare vary based on day of week and weather conditions. It:

1. Creates a CTE (Common Table Expression) to identify all trips from the Loop to O'Hare, extracting day of week and hour
2. Joins this with weather data, categorizing conditions as "Bad" or "Good"
3. Categorizes trips into four groups: Rainy Saturday, Non-Rainy Saturday, Rainy Other Day, and Non-Rainy Other Day
4. Calculates average duration for each category in both seconds and minutes
5. Orders results by duration to highlight the longest average rides

Best Practices:

- Using a CTE improves readability by breaking the complex query into logical components
- `EXTRACT` function efficiently pulls specific date parts (DOW, HOUR) for analysis
- The categorization approach with `CASE` statements creates meaningful comparison groups
- Including both seconds and minutes in the output improves readability for different audiences
- The nested query for weather categorization maintains consistency with Query 5's approach

---