fingerTips

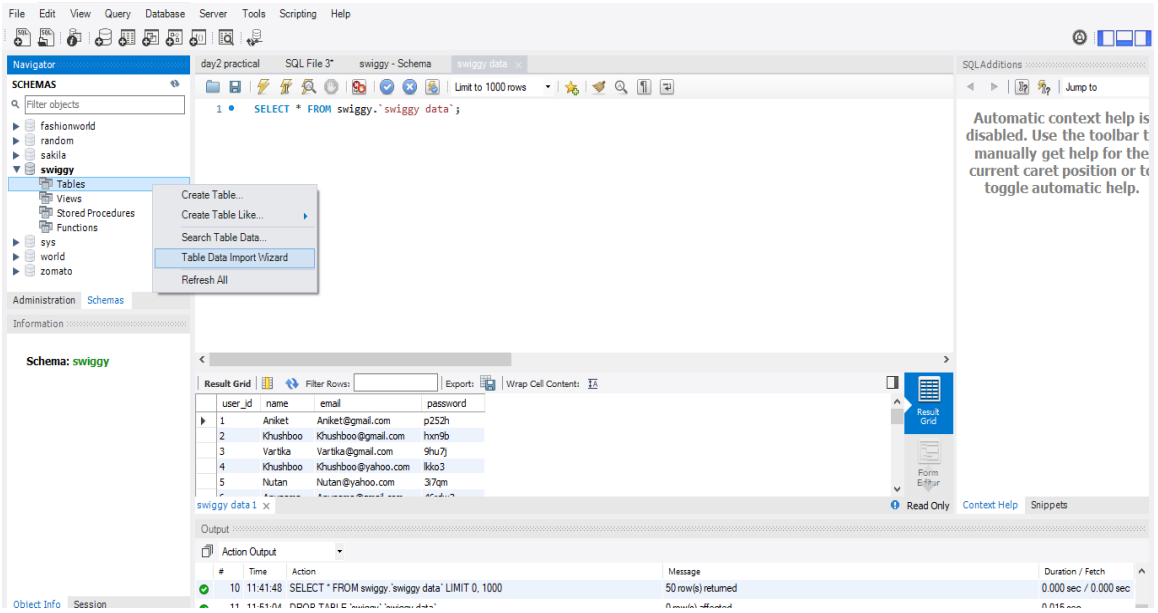
Module-6

Ranking & Analytical Functions in SQL

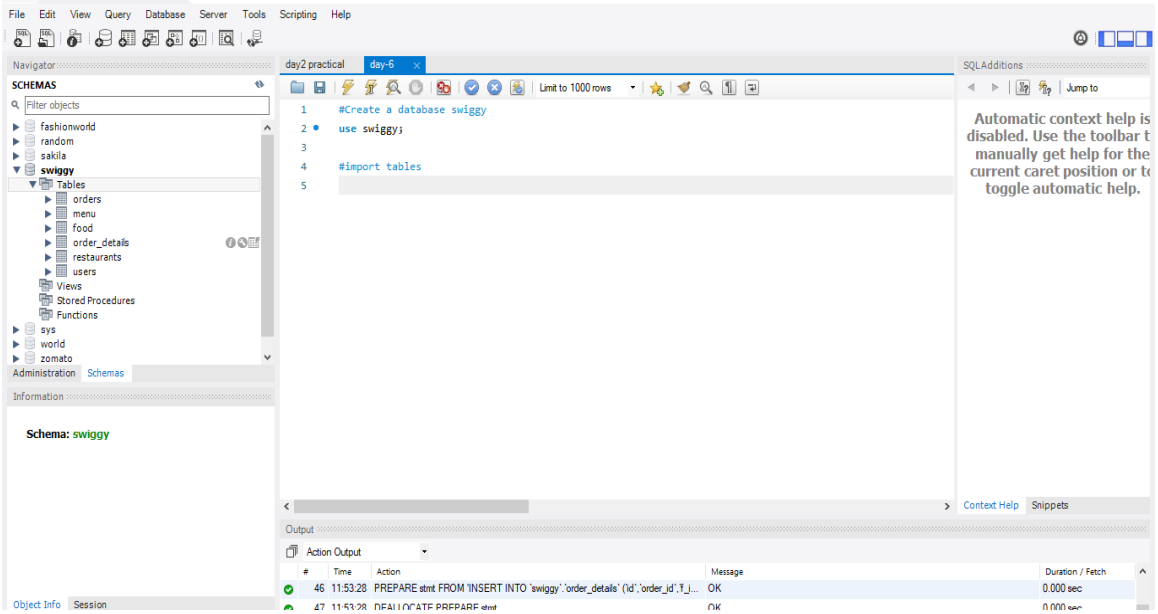
1) Introduction to Case Study-2, Swiggy dataset

We have 6 tables in Swiggy database:

i. Users
ii. Restaurants
iii. Food
iv. Menu
v. Orders
vi. Order_details



And one by one we will import all the 6 tables of Swiggy database.



And we will now answer some questions using Ranking & Analytic functions in SQL.

2) Ranking Functions in SQL (Row number, Rank, Dense rank)

Rank, dense rank, and row number are three common ranking functions used in SQL to assign a unique rank to each row in a result set based on a specific column or set of columns. While they may appear similar, there are important differences between them that can affect their behavior in different scenarios.

i. ROW_NUMBER:

ROW_NUMBER function is used to generate a unique number for each row in a result set. The generated numbers are consecutive integers that start at 1 for the first row and increment by 1 for each subsequent row. This function is useful when you need to generate a unique identifier for each row, regardless of the values in any other columns.

Syntax:

SELECT ROW_NUMBER() OVER(column_name ORDER BY column_name) AS row_num, column_name FROM table_name;

ii. RANK:

The RANK function assigns a unique rank to each distinct value in a result set based on a specific column or set of columns. If two or more rows have the same value in the ranking column(s), they will be assigned the same rank, and the next rank will be skipped. For example, if three rows have a value of 10 in the ranking column, they will be assigned rank 1, and the next rank will be 4.

Syntax:

SELECT RANK() OVER(ORDER BY column_name) AS rank_num, column_name FROM table_name;

iii. DENSE_RANK:

The DENSE_RANK function is similar to the RANK function in that it assigns a unique rank to each distinct value in a result set based on a specific column or set of columns. However, if two or more rows have the same value in the ranking column(s), they will be assigned the same rank, and the next rank will be the next consecutive integer. For example, if three rows have a value of 10 in the ranking column, they will be assigned rank 1, and the next rank will be 2.

Syntax:

SELECT DENSE_RANK() OVER(ORDER BY column_name) AS dense_rank_num, column_name FROM table_name;

The main difference between RANK and DENSE_RANK is that the RANK function can skip ranks, while the DENSE_RANK function does not. In addition, ROW_NUMBER function generates unique consecutive integers for each row, whereas both RANK and DENSE_RANK generate unique rankings for each distinct value in a column or set of columns.

ROW_NUMBER generates unique numbers for each row, while RANK and DENSE_RANK assign a unique rank to each distinct value in a column or set of columns. RANK can skip ranks, while DENSE_RANK does not.

Example:

Q. What are the delivery ratings given by each user, arrange them in descending order?

Row number

```
select u.name, o.delivery_rating,
row_number() over(partition by u.name order by
o.delivery_rating desc) as rank_rating
from users1 u join orders1 o
on u.user_id=o.user_id;
```

Rank

```
select u.name, o.delivery_rating,
rank() over(partition by u.name order by o.delivery_rating desc) as rank_rating
from users1 u join orders1 o
on u.user_id=o.user_id;
```

Dense rank

```
select u.name, o.delivery_rating,
dense_rank() over(partition by u.name order by o.delivery_rating desc) as rank_rating
from users1 u join orders1 o
on u.user_id=o.user_id;
```

Q. What is the amount of food ordered by each user, arrange in descending order with respect to amount?

Row number

```
select u.name, o.amount,
row_number() over(partition by u.name order by o.amount desc) as rank_amount
from users1 u join orders1 o
on u.user_id=o.user_id;
```

Rank

```
select u.name, o.amount,
rank() over(partition by u.name order by o.amount desc) as rank_amount
from users1 u join orders1 o
on u.user_id=o.user_id;
```

Dense rank

```
select u.name, o.amount,
```

```
dense_rank() over(partition by u.name order by o.amount desc) as rank_amount
from users1 u join orders1 o
on u.user_id=o.user_id;
```

Q. Which cuisine is sold for highest price?

Row number

```
select r.cuisine, m.price,
row_number() over(partition by  r.cuisine order by r.cuisine, m.price desc) as rank_price
from restaurants_1 r join menu_1 m
on r.r_id=m.r_id;
```

Rank

```
select r.cuisine, m.price,
rank() over(partition by  r.cuisine order by r.cuisine, m.price desc) as rank_price
from restaurants_1 r join menu_1 m
on r.r_id=m.r_id;
```

Dense rank

```
select r.cuisine, m.price,
dense_rank() over(partition by  r.cuisine order by r.cuisine, m.price desc) as rank_price
from restaurants_1 r join menu_1 m
on r.r_id=m.r_id;
```

3) Analytic Functions in SQL (Lag, Lead)

In SQL, lead and lag are analytical functions that operate on a specified column in a result set. These functions are used to access the value of a column in the current row relative to the value of the same column in a previous or subsequent row.

The lead function retrieves the value of a column in the next row, while the lag function retrieves the value of a column in the

previous row. Both functions take three arguments: the column to be retrieved, the number of rows to shift, and the default value to be returned if there is no next or previous row.

Syntax for lag():
SELECT id, value, lag(value) OVER (ORDER BY id) AS prev_value
FROM my_table;

Syntax for lead():
SELECT id, value, lead(value) OVER (ORDER BY id) AS next_value
FROM my_table;

Overall, lead and lag functions in SQL are useful for analyzing the relationship between values in a column across multiple rows. They can be used for a variety of purposes, such as detecting trends, identifying outliers, and calculating moving averages.

Examples:
Lag:
Q. For each person find whether he has spent more/less on food than previous day.

```
select u.name, o.date, o.amount,
lag(amount) over(partition by u.name order by o.date) as previous_amount
from orders1 o join users1 u
on o.user_id=u.user_id;
```

Lead:
Q. For each person find whether he has spent more/less on food than next day.

```
select u.name, o.date, o.amount,
lead(amount) over(partition by u.name order by o.date) as previous_amount
from orders1 o join users1 u
on o.user_id=u.user_id;
```