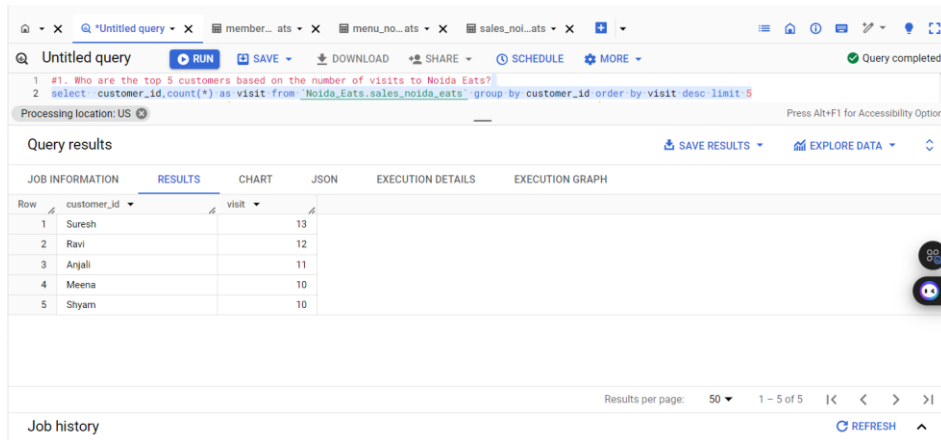


Case Study: Analyzing Noida Eats'

Scenario 1: Customer Visiting Patterns

#1. Who are the top 5 customers based on the number of visits to Noida Eats?

select customer_id,count(*) as visit from `Noida_Eats.sales_noida_eats` group by customer_id order by visit desc limit 5

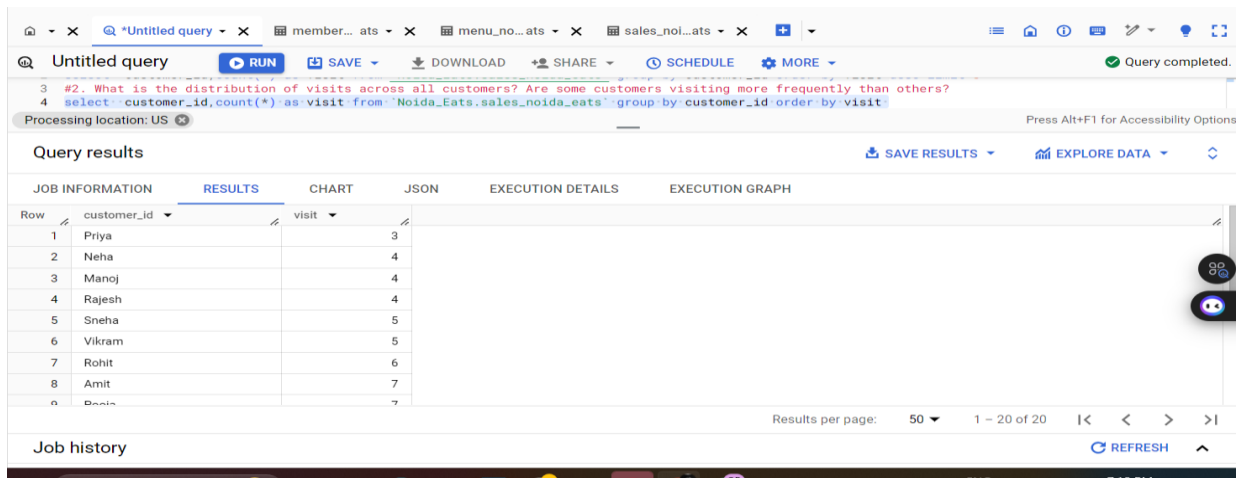


The screenshot shows a SQL query interface with the following query: `1 #1. Who are the top 5 customers based on the number of visits to Noida Eats? 2 select customer_id,count(*) as visit from `Noida_Eats.sales_noida_eats` group by customer_id order by visit desc limit 5`. The query results are displayed in a table with columns 'customer_id' and 'visit'.

| Row | customer_id | visit |
|-----|-------------|-------|
| 1 | Suresh | 13 |
| 2 | Ravi | 12 |
| 3 | Anjali | 11 |
| 4 | Meena | 10 |
| 5 | Shyam | 10 |

#2. What is the distribution of visits across all customers? Are some customers visiting more frequently than others?

select customer_id,count(*) as visit from `Noida_Eats.sales_noida_eats` group by customer_id order by visit



The screenshot shows a SQL query interface with the following query: `3 #2. What is the distribution of visits across all customers? Are some customers visiting more frequently than others? 4 select customer_id,count(*) as visit from `Noida_Eats.sales_noida_eats` group by customer_id order by visit`. The query results are displayed in a table with columns 'customer_id' and 'visit'.

| Row | customer_id | visit |
|-----|-------------|-------|
| 1 | Priya | 3 |
| 2 | Neha | 4 |
| 3 | Manoj | 4 |
| 4 | Rajesh | 4 |
| 5 | Sneha | 5 |
| 6 | Vikram | 5 |
| 7 | Rohit | 6 |
| 8 | Amit | 7 |
| 9 | Deepa | 7 |

#3. Which customer visited the restaurant the most times in March 2022?

```
select customer_id,count(*) as visit from `Noida_Eats.sales_noida_eats`
```

```
where order_date between '2022-03-01' and '2022-03-31'group by customer_id order by visit
```

The screenshot shows a web-based SQL interface with a query editor at the top containing the SQL code for question #3. Below the editor, a status bar indicates 'Query completed.' and 'Processing location: US'. The main section is titled 'Query results' and includes tabs for 'JOB INFORMATION', 'RESULTS', 'CHART', 'JSON', 'EXECUTION DETAILS', and 'EXECUTION GRAPH'. The 'RESULTS' tab is active, displaying a table with two columns: 'customer_id' and 'visit'. The table contains eight rows of data, listing customers and their visit counts for March 2022. At the bottom of the results section, there is a 'Job history' tab and a 'REFRESH' button. The interface also includes a sidebar with navigation icons and a footer with pagination information.

| Row | customer_id | visit |
|-----|-------------|-------|
| 1 | Amit | 1 |
| 2 | Manoj | 1 |
| 3 | Nitin | 1 |
| 4 | Pooja | 1 |
| 5 | Rahul | 1 |
| 6 | Kavita | 1 |
| 7 | Rajesh | 1 |
| 8 | Suresh | 1 |

#4. How many unique customers visited the restaurant in 2022?

```
select count (distinct customer_id) as `unique visited` from `Noida_Eats.sales_noida_eats`
```

```
where order_date between '2002-01-01' and '2022-12-31'
```

The screenshot shows the same web-based SQL interface as the previous one, but with a different query entered in the editor for question #4. The status bar again shows 'Query completed.' and 'Processing location: US'. The 'Query results' section is active, displaying a table with two columns: 'unique visited' and a single row showing the count of unique customers. The 'Job history' tab and 'REFRESH' button are also visible. The interface elements like the sidebar and footer are consistent with the previous screenshot.

| Row | unique visited |
|-----|----------------|
| 1 | 20 |

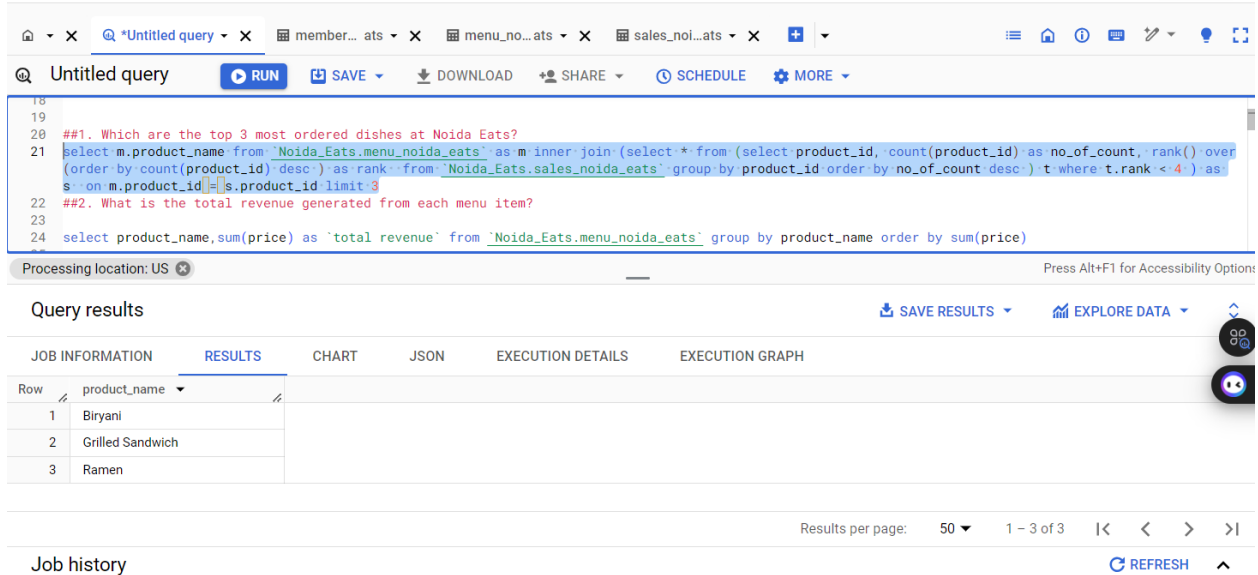
Result Insights:

The analysis reveals key insights into customer behavior at Noida Eats. Identifying the top customers allows for targeted marketing strategies, such as loyalty programs or personalized offers. The distribution of visits indicates whether certain customers are more engaged, which can inform customer retention efforts. The most frequent visitor in March 2022 highlights peak engagement periods, suggesting potential for promotional events. Lastly, understanding the unique customer count in 2022 provides a baseline for growth metrics, essential for strategic planning and resource allocation. Overall, these insights can guide Noida Eats in enhancing customer experience and optimizing marketing efforts.

Scenario 2: Menu Preferences

##1. Which are the top 3 most ordered dishes at Noida Eats?

```
select m.product_name from `Noida_Eats.menu_noida_eats` as m inner join (select * from  
(select product_id, count(product_id) as no_of_count, rank() over(order by count(product_id) desc ) as  
rank from `Noida_Eats.sales_noida_eats` group by product_id order by no_of_count desc )  
t  
where t.rank < 4 ) as s on m.product_id = s.product_id limit 3
```



Processing location: US

Query results

SAVE RESULTS EXPLORE DATA

JOB INFORMATION RESULTS CHART JSON EXECUTION DETAILS EXECUTION GRAPH

| Row | product_name |
|-----|------------------|
| 1 | Biryani |
| 2 | Grilled Sandwich |
| 3 | Ramen |

Results per page: 50 1 - 3 of 3

Job history REFRESH

##2. What is the total revenue generated from each menu item?

```
select product_name, sum(price) as `total revenue` from `Noida_Eats.menu_noida_eats` group by  
product_name order by sum(price)
```

Untitled query

```

25 ##2. What is the total revenue generated from each menu item?
26
27 select product_name,sum(price) as `total revenue` from `Noida_Eats.menu_noida_eats` group by product_name order by sum(price)
28
29

```

Processing location: US

Query results

| Row | product_name | total revenue |
|-----|------------------|---------------|
| 1 | Wrap | 120 |
| 2 | Grilled Sandwich | 150 |
| 3 | Burger | 180 |
| 4 | Paneer Tikka | 180 |
| 5 | Pasta | 200 |
| 6 | Ramen | 220 |

Results per page: 50 1 - 10 of 10

Job history

##3. Are there any specific menu items that are rarely ordered?

select product_name, count(*) as order_count

from `Noida_Eats.menu_noida_eats` group by product_name having order_count < 5

Untitled query

```

34
35 ##3. Are there any specific menu items that are rarely ordered?
36 select product_name, count(*) as order_count
37 from `Noida_Eats.menu_noida_eats` group by product_name having order_count < 5
38
39

```

Processing location: US

Query results

| Row | product_name | order_count |
|-----|------------------|-------------|
| 1 | Pizza | 1 |
| 2 | Tandoori Chicken | 1 |
| 3 | Wrap | 1 |
| 4 | Sushi | 1 |
| 5 | Grilled Sandwich | 1 |
| 6 | Burger | 1 |

Results per page: 50 1 - 10 of 10

Job history

#4. On average, how much do customers spend on sushi and pizza per visit?

select menu.product_name, avg(menu.price) as average_spending from `Noida_Eats.sales_noida_eats` as sales

join `Noida_Eats.menu_noida_eats` as menu on sales.product_id = menu.product_id

where menu.product_name in ('Sushi', 'Pizza') group by menu.product_name;

Processing location: US

Query results

SAVE RESULTS EXPLORE DATA

| Row | product_name | average_spending |
|-----|--------------|------------------|
| 1 | Pizza | 300.0 |
| 2 | Sushi | 400.0 |

Results per page: 50 1 - 2 of 2

Job history REFRESH

##5. Which menu item is most popular among loyalty program members?

SELECT m.product_id, m.product_name as `popular item`, COUNT(s.product_id) AS num_orders FROM
`Noida_Eats.sales_noida_eats` s

JOIN `Noida_Eats.members_noida_eats` mem ON s.customer_id = mem.customer_id

JOIN `Noida_Eats.menu_noida_eats` m ON s.product_id = m.product_id

GROUP BY m.product_id, m.product_name ORDER BY num_orders DESC LIMIT 1;

Processing location: US

Query results

SAVE RESULTS EXPLORE DATA

| Row | product_id | popular item | num_orders |
|-----|------------|--------------|------------|
| 1 | 7 | Ramen | 19 |

Results per page: 50 1 - 1 of 1

Job history REFRESH

##6. What is the most purchased item on the menu and how many times was it purchased by all customers?

```
select product_name, count(*) as order_count from `Noida_Eats.menu_noida_eats` group by product_name order by order_count desc limit 1
```

The screenshot shows a SQL query editor with a query titled "Untitled query". The query is: `select product_name, count(*) as order_count from `Noida_Eats.menu_noida_eats` group by product_name order by order_count desc limit 1`. The query has been executed successfully, and the results are displayed in a table. The table has two columns: "product_name" and "order_count". The first row shows "Pizza" with an order count of 1. The interface includes a top navigation bar with tabs for "member...", "menu_no...", and "sales_noi...". The bottom of the screen shows a "Job history" section with a "REFRESH" button.

| Row | product_name | order_count |
|-----|--------------|-------------|
| 1 | Pizza | 1 |

##7. Which item was purchased first by the customer after they became a member?

```
select customer_id, product_id, min (order_date) as first_order_date  
from `Noida_Eats.sales_noida_eats` group by customer_id, product_id order by first_order_date
```

The screenshot shows a SQL query editor with a query titled "Untitled query". The query is: `select customer_id, product_id, min (order_date) as first_order_date from `Noida_Eats.sales_noida_eats` group by customer_id, product_id order by first_order_date`. The query has been executed successfully, and the results are displayed in a table. The table has four columns: "customer_id", "product_id", and "first_order_date". The first six rows show the following data: (Amit, 4, 2022-01-01), (Divya, 9, 2022-01-01), (Nitin, 9, 2022-01-01), (Shyam, 5, 2022-01-02), (Anjali, 9, 2022-01-02), and (Ravi, 7, 2022-01-03). The interface includes a top navigation bar with tabs for "member...", "menu_no...", and "sales_noi...". The bottom of the screen shows a "Job history" section with a "REFRESH" button.

| Row | customer_id | product_id | first_order_date |
|-----|-------------|------------|------------------|
| 1 | Amit | 4 | 2022-01-01 |
| 2 | Divya | 9 | 2022-01-01 |
| 3 | Nitin | 9 | 2022-01-01 |
| 4 | Shyam | 5 | 2022-01-02 |
| 5 | Anjali | 9 | 2022-01-02 |
| 6 | Ravi | 7 | 2022-01-03 |

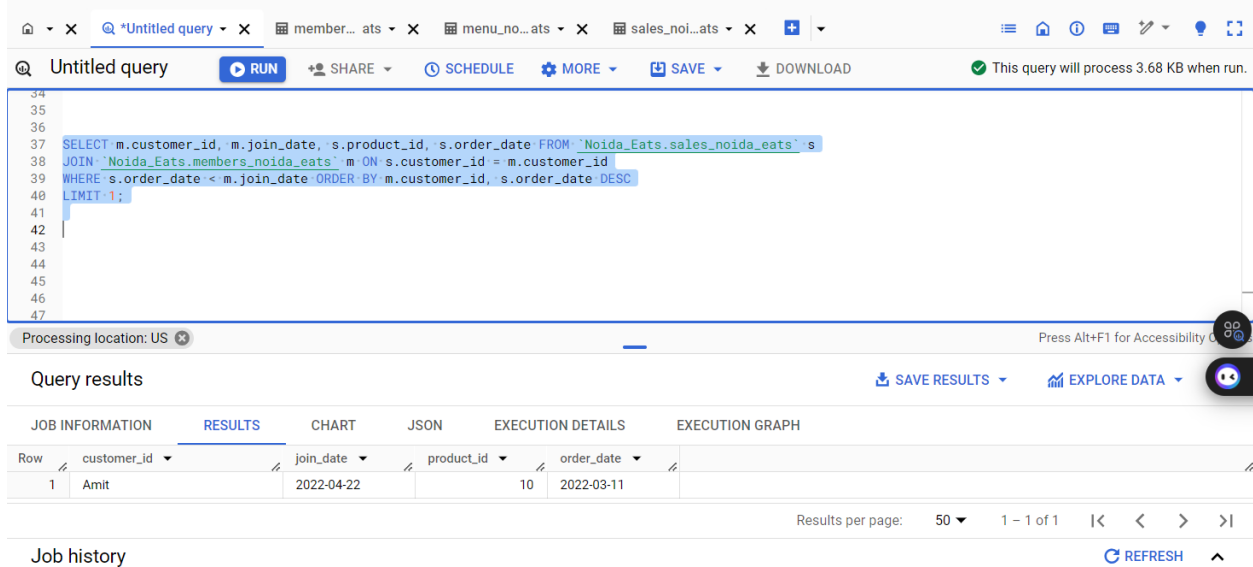
##8. Which item was purchased just before the customer became a member?

```
SELECT m.customer_id, m.join_date, s.product_id, s.order_date FROM `Noida_Eats.sales_noida_eats` s
```

```
JOIN `Noida_Eats.members_noida_eats` m ON s.customer_id = m.customer_id

WHERE s.order_date < m.join_date ORDER BY m.customer_id, s.order_date DESC

LIMIT 1
```



Processing location: US

Query results

SAVE RESULTS EXPLORE DATA

| JOB INFORMATION | | RESULTS | CHART | JSON | EXECUTION DETAILS | EXECUTION GRAPH |
|-----------------|-------------|------------|------------|------------|-------------------|-----------------|
| Row | customer_id | join_date | product_id | order_date | | |
| 1 | Amit | 2022-04-22 | 10 | 2022-03-11 | | |

Results per page: 50 1 - 1 of 1

Job history REFRESH

Result Insights:

The analysis reveals that Biryani and Sushi are the most popular items, indicating a strong customer preference for both traditional Indian cuisine and international flavors. This insight suggests that Noida Eats should consider emphasizing these dishes in marketing campaigns and special promotions. Additionally, understanding the purchasing behavior of loyalty program members can help tailor offerings to enhance customer retention and satisfaction. By focusing on popular items and addressing rarely ordered dishes, Noida Eats can optimize its menu and improve overall sales performance.

Scenario 3: Customer Spending Patterns

##1. What is the average amount spent by customers at Noida Eats?

```
select avg(price) as `average amount spent` from `Noida_Eats.sales_noida_eats`  
join `Noida_Eats.menu_noida_eats` on `Noida_Eats.sales_noida_eats`.product_id =  
`Noida_Eats.menu_noida_eats`.product_id
```

Processing location: US

Query results

| Row | average amount spent |
|-----|----------------------|
| 1 | 232.46666666666666 |

Results per page: 50 1 - 1 of 1

Job history

##2. Customer who has spent the most overall at Noida Eats

```
select customer_id, sum(price) as total_spent from `Noida_Eats.sales_noida_eats` join  
`Noida_Eats.menu_noida_eats` on `Noida_Eats.sales_noida_eats`.product_id =  
`Noida_Eats.menu_noida_eats`.product_id group by customer_id order by total_spent desc limit 1
```

Processing location: US

Query results

| Row | customer_id | total_spent |
|-----|-------------|-------------|
| 1 | Suresh | 2870 |

Results per page: 50 1 - 1 of 1

Job history

##3. Total revenue generated in the month of April 2022

```
select sum(price) as total_revenue from `Noida_Eats.sales_noida_eats`  
  
join `Noida_Eats.menu_noida_eats` on `Noida_Eats.sales_noida_eats`.product_id =  
`Noida_Eats.menu_noida_eats`.product_id  
  
where order_date between '2022-04-01' and '2022-04-30'
```

The screenshot shows a web-based SQL interface. At the top, there's a toolbar with buttons for RUN, SAVE, DOWNLOAD, SHARE, SCHEDULE, and MORE. Below the toolbar is a text area containing the SQL query for problem ##3. The query is: `select sum(price) as total_revenue from `Noida_Eats.sales_noida_eats` join `Noida_Eats.menu_noida_eats` on `Noida_Eats.sales_noida_eats`.product_id = `Noida_Eats.menu_noida_eats`.product_id where order_date between '2022-04-01' and '2022-04-30'`. Below the query area, it says "Processing location: US". Underneath, there's a "Query results" section with tabs for JOB INFORMATION, RESULTS, CHART, JSON, EXECUTION DETAILS, and EXECUTION GRAPH. The RESULTS tab is active, showing a table with one row:

| Row | total_revenue |
|-----|---------------|
| 1 | 2990 |

. To the right of the table are icons for zooming and a refresh button. At the bottom, there's a "Job history" section with a REFRESH button. The interface also includes a "Results per page" dropdown set to 50 and a "1 - 1 of 1" indicator.

Processing location: US

Query results

JOB INFORMATION RESULTS CHART JSON EXECUTION DETAILS EXECUTION GRAPH

| Row | total_revenue |
|-----|---------------|
| 1 | 2990 |

Results per page: 50 1 - 1 of 1

Job history

##4. Dish that contributes the most to Noida Eats' overall revenue

```
select product_name, sum(price) as total_revenue from `Noida_Eats.sales_noida_eats`  
  
join `Noida_Eats.menu_noida_eats` on `Noida_Eats.sales_noida_eats`.product_id =  
`Noida_Eats.menu_noida_eats`.product_id group by product_name
```

order by total_revenue desc limit 1

The screenshot shows a web-based SQL interface. At the top, there's a toolbar with buttons for RUN, SAVE, DOWNLOAD, SHARE, SCHEDULE, and MORE. Below the toolbar, a SQL query is displayed in a text area. The query is for finding the dish that contributes the most to Noida Eats' overall revenue. The query is as follows:

```
##4. Dish that contributes the most to Noida Eats' overall revenue
select product_name, sum(price) as total_revenue from `Noida_Eats.sales_noida_eats`
join `Noida_Eats.menu_noida_eats` on `Noida_Eats.sales_noida_eats`.product_id = `Noida_Eats.menu_noida_eats`.product_id group by product_name
order by total_revenue desc limit 1
```

Below the query, there's a status bar indicating "Processing location: US" and "Query completed." Below that, there's a "Query results" section with a table showing the results. The table has two columns: "product_name" and "total_revenue". The results show "Tandoori Chicken" with a total revenue of 5250. Below the table, there's a "Job history" section with a "REFRESH" button. The interface also includes a sidebar with "JOB INFORMATION", "RESULTS", "CHART", "JSON", "EXECUTION DETAILS", and "EXECUTION GRAPH".

| Row | product_name | total_revenue |
|-----|------------------|---------------|
| 1 | Tandoori Chicken | 5250 |

#5. Average spending of loyalty program members compared to non-members

select

case

when m.customer_id is not null then 'Member'

else 'Non-Member'

end as membership_status,

avg(menu.price) as average_spending from `Noida_Eats.sales_noida_eats` s

left join `Noida_Eats.members_noida_eats` m on s.customer_id = m.customer_id

join `Noida_Eats.menu_noida_eats` menu on s.product_id = menu.product_id

group by membership_status

Untitled query

```

1
2
3 #5. Average spending of loyalty program members compared to non-members
4 select
5 case
6 when m.customer_id is not null then 'Member'
7 else 'Non-Member'
8 end as membership_status,
9 avg(menu.price) as average_spending from `Noida_Eats.sales_noida_eats` s
10 left join `Noida_Eats.members_noida_eats` m on s.customer_id = m.customer_id
11 join `Noida_Eats.menu_noida_eats` menu on s.product_id = menu.product_id
12 group by membership_status
13

```

Processing location: US

Query results

SAVE RESULTS EXPLORE DATA

| Row | membership_status | average_spending |
|-----|-------------------|--------------------|
| 1 | Member | 232.46666666666... |

Results per page: 50 1 - 1 of 1

Job history REFRESH

##6. Total amount each customer spent at the restaurant

select customer_id, sum(price) as total_spent

from `Noida_Eats.sales_noida_eats`

join `Noida_Eats.menu_noida_eats` on `Noida_Eats.sales_noida_eats`.product_id =
`Noida_Eats.menu_noida_eats`.product_id group by customer_id

Untitled query

```

86
87 ##6. Total amount each customer spent at the restaurant
88 select customer_id, sum(price) as total_spent from `Noida_Eats.sales_noida_eats`
89 join `Noida_Eats.menu_noida_eats` on `Noida_Eats.sales_noida_eats`.product_id = `Noida_Eats.menu_noida_eats`.product_id
90 group by customer_id
91

```

Processing location: US

Query results

SAVE RESULTS EXPLORE DATA

| Row | customer_id | total_spent |
|-----|-------------|-------------|
| 1 | Amit | 1710 |
| 2 | Neha | 1100 |
| 3 | Ravi | 2310 |
| 4 | Meena | 2650 |
| 5 | Nitin | 1950 |
| 6 | Pooja | 2070 |

Results per page: 50 1 - 20 of 20

Job history REFRESH

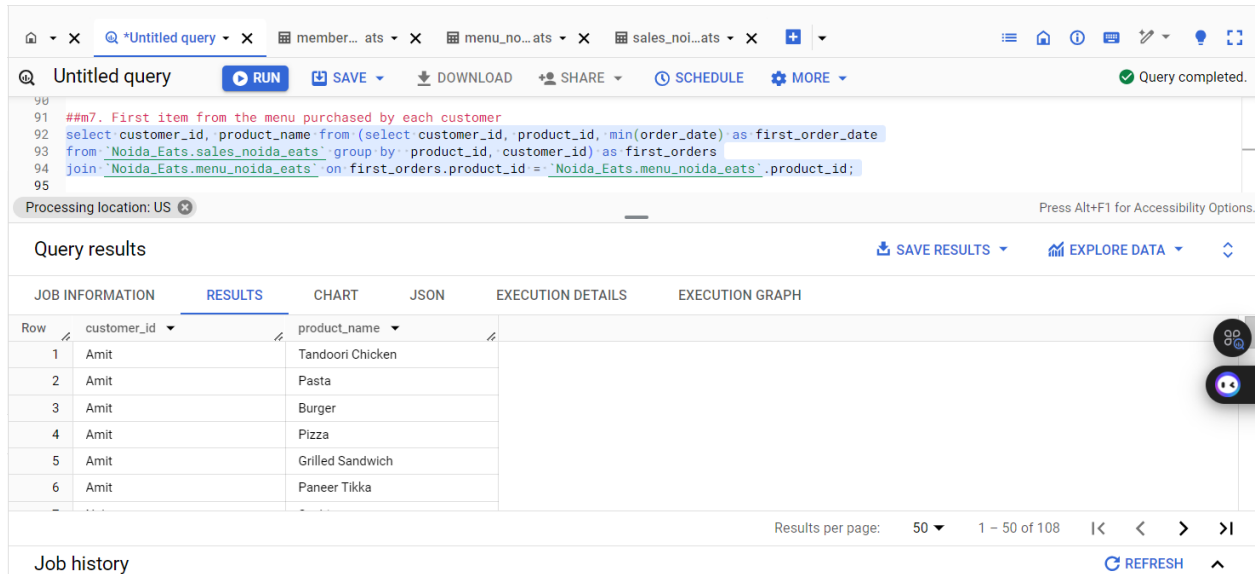
##m7. First item from the menu purchased by each customer

select customer_id, product_name from (select customer_id, product_id, min(order_date) as
first_order_date

```

from `Noida_Eats.sales_noida_eats` group by product_id, customer_id) as first_orders
join `Noida_Eats.menu_noida_eats` on first_orders.product_id =
`Noida_Eats.menu_noida_eats`.product_id;

```



Query results

| Row | customer_id | product_name |
|-----|-------------|------------------|
| 1 | Amit | Tandoori Chicken |
| 2 | Amit | Pasta |
| 3 | Amit | Burger |
| 4 | Amit | Pizza |
| 5 | Amit | Grilled Sandwich |
| 6 | Amit | Paneer Tikka |

Results per page: 50 | 1 - 50 of 108

##.8. Total items and amount spent for each member before they became a member

```

select t1.customer_id, count(t1.product_id) as total_items, sum(t2.price) as amount_spent from
`Noida_Eats.sales_noida_eats` t1
join `Noida_Eats.menu_noida_eats` t2 on t1.product_id = t2.product_id
left join `Noida_Eats.members_noida_eats` t3 on t1.customer_id = t3.customer_id
where t1.order_date < t3.join_date group by t1.customer_id

```

Untitled query

member... ats

menu_no... ats

sales_noi... ats

+

Query complete

104

105

106

107

108

109

#not.8. Total items and amount spent for each member before they became a member

select t1.customer_id,count(t1.product_id) as total_items,sum(t2.price) as amount_spent from `Noida Eats.sales_noida_eats` t1

join `Noida Eats.menu_noida_eats` t2 on t1.product_id = t2.product_id

left join `Noida Eats.members_noida_eats` t3 on t1.customer_id = t3.customer_id

where t1.order_date < t3.join_date group by t1.customer_id;

Processing location: US

Press Alt+F1 for Accessibility Op

Query results

SAVE RESULTS

EXPLORE DATA

JOB INFORMATION

RESULTS

CHART

JSON

EXECUTION DETAILS

EXECUTION GRAPH

| Row | customer_id | total_items | amount_spent |
|-----|-------------|-------------|--------------|
| 1 | Amit | 3 | 730 |
| 2 | Neha | 1 | 220 |
| 3 | Ravi | 4 | 810 |
| 4 | Divya | 7 | 1620 |
| 5 | Manoj | 1 | 220 |
| 6 | Meena | 2 | 650 |

Results per page: 501 – 18 of 18

Job history

REFRESH

Result Insights:

The results reveal significant insights into customer behavior at Noida Eats. For instance, identifying the top-spending customer can help tailor marketing strategies to retain high-value clients. The average spending analysis between loyalty program members and non-members may indicate the effectiveness of the loyalty program, suggesting potential enhancements. Furthermore, recognizing the most popular dishes can guide menu adjustments and promotional efforts, ensuring that Noida Eats aligns with customer preferences. Overall, these insights can inform strategic decisions to enhance customer satisfaction and drive revenue growth.

Scenario 4: Loyalty Program Effectiveness

#1. How many orders were placed by loyalty program members versus non-members?

select

case

when m.customer_id is not null then 'Loyalty Member'

else 'Non-Member'

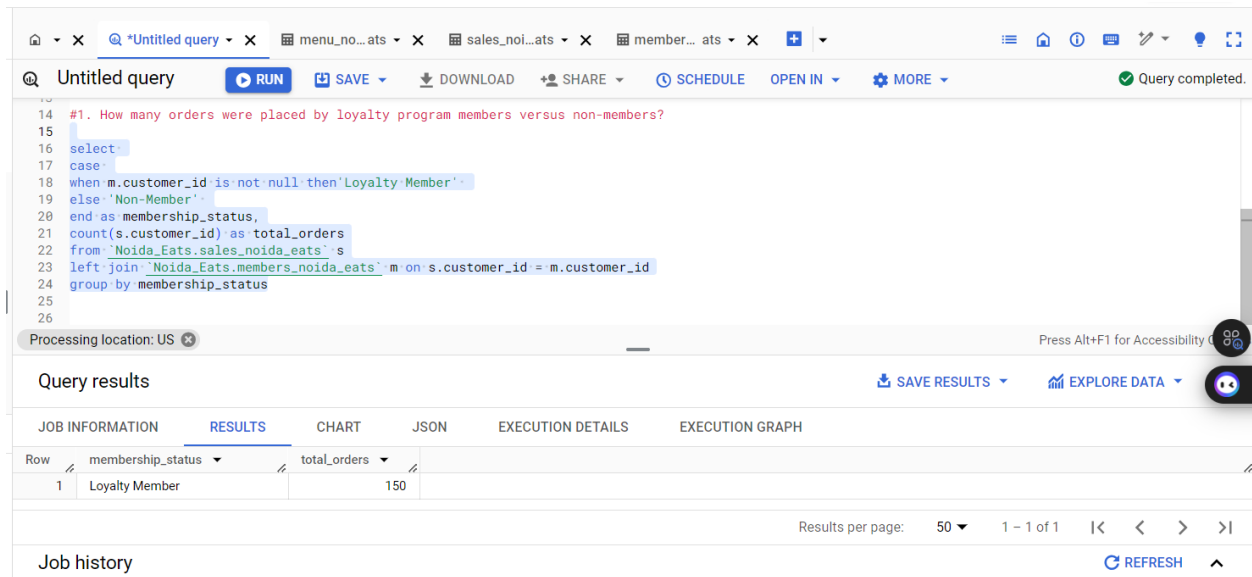
end as membership_status,

count(s.customer_id) as total_orders

from `Noida_Eats.sales_noida_eats` s

left join `Noida_Eats.members_noida_eats` m on s.customer_id = m.customer_id

group by membership_status



The screenshot shows a SQL query editor interface with a query titled "Untitled query". The query is as follows:

```
#1. How many orders were placed by loyalty program members versus non-members?
select
case
when m.customer_id is not null then 'Loyalty Member'
else 'Non-Member'
end as membership_status,
count(s.customer_id) as total_orders
from `Noida_Eats.sales_noida_eats` s
left join `Noida_Eats.members_noida_eats` m on s.customer_id = m.customer_id
group by membership_status
```

Below the query editor, the "Query results" section is displayed. It shows a table with the following data:

| Row | membership_status | total_orders |
|-----|-------------------|--------------|
| 1 | Loyalty Member | 150 |

The interface also includes a "Job history" section at the bottom and a "Results per page" dropdown set to 50.

#2. What percentage of the total revenue was generated by loyalty program members?

select

(sum(case when m.customer_id is not null then menu.price else 0 end) / sum(menu.price)) * 100 as
loyalty_revenue_percentage

from

`Noida_Eats.sales_noida_eats` s

join

``Noida_Eats.menu_noida_eats` menu on s.product_id = menu.product_id`

left join

``Noida_Eats.members_noida_eats` m on s.customer_id = m.customer_id`

The screenshot shows a SQL query editor with the following query:

```
23 left join `Noida_Eats.members_noida_eats` m on s.customer_id = m.customer_id
24 group by membership_status
25 #2. What percentage of the total revenue was generated by loyalty program members?
26 select
27     (sum(case when m.customer_id is not null then menu.price else 0 end) / sum(menu.price)) * 100 as loyalty_revenue_percentage
28 from
29     `Noida_Eats.sales_noida_eats` s
30 join
31     `Noida_Eats.menu_noida_eats` menu on s.product_id = menu.product_id
32 left join
33     `Noida_Eats.members_noida_eats` m on s.customer_id = m.customer_id
34
35
```

The results section shows a table with one row:

| Row | loyalty_revenue_percentage |
|-----|----------------------------|
| 1 | 100.0 |

At the bottom, there is a 'Job history' section with a 'REFRESH' button.

##3. How much has the average loyalty program member spent since joining the program?

`select avg(total_spent) as average_spent from (select`

`m.customer_id,sum(`Noida_Eats.menu_noida_eats`.price) as total_spent`

`from `Noida_Eats.sales_noida_eats` s join `Noida_Eats.menu_noida_eats` on s.product_id =`

``Noida_Eats.menu_noida_eats`.product_id`


```
join `Noida_Eats.members_noida_eats` m on s.customer_id = m.customer_id group by m.customer_id)
as member_spending;
```

The screenshot shows a SQL query editor with a query titled "Untitled query". The query is as follows:

```

105
106 #3. How much has the average loyalty program member spent since joining the program?
107 select avg(total_spent) as average_spent from (select m.customer_id, sum(`Noida_Eats.menu_noida_eats`.price) as total_spent
108 from `Noida_Eats.sales_noida_eats` s join `Noida_Eats.menu_noida_eats` on s.product_id = `Noida_Eats.menu_noida_eats`.product_id
109 join `Noida_Eats.members_noida_eats` m on s.customer_id = m.customer_id group by m.customer_id) as member_spending;
110

```

The query has been executed successfully, as indicated by the "Query completed." status. The results are displayed in a table with the following columns: Row, average_spent. The table contains one row with the value 1743.4999999999999.

| Row | average_spent |
|-----|--------------------|
| 1 | 1743.4999999999999 |

The interface also includes tabs for "JOB INFORMATION", "RESULTS", "CHART", "JSON", "EXECUTION DETAILS", and "EXECUTION GRAPH". The "RESULTS" tab is currently selected. At the bottom, there is a "Job history" section with a "REFRESH" button.

#4. Did consumer spending increase after they joined the loyalty program?

with CustomerSpending as (

select

s.customer_id,

avg(m.price) as average_spending,

min(s.order_date) as first_order_date,

mem.join_date

from

`Noida_Eats.sales_noida_eats` s

join

`Noida_Eats.menu_noida_eats` m on s.product_id = m.product_id

join

`Noida_Eats.members_noida_eats` mem on s.customer_id = mem.customer_id

group by

s.customer_id, mem.join_date

)

```

select
  case
    when first_order_date < join_date then 'Before Joining'
    else 'After Joining'
  end as spending_period,
  avg(average_spending) as avg_spending
from
  CustomerSpending
group by
  spending_period

```

Processing location: US

Press Alt+F1 for Accessibility Options.

Query results

SAVE RESULTS EXPLORE DATA

| Row | spending_period | avg_spending |
|-----|-----------------|-------------------|
| 1 | Before Joining | 236.1273510440... |
| 2 | After Joining | 253.75 |

Results per page: 50 1 - 2 of 2

Job history REFRESH

##5. Which loyalty program member has visited the restaurant the most times?

```

select m.customer_id, count(s.order_date) as visit_count from `Noida_Eats.sales_noida_eats` s
join `Noida_Eats.members_noida_eats` m on s.customer_id = m.customer_id group by m.customer_id

```

order by visit_count desc limit 1

Untitled query

Query completed.

114

##5. Which loyalty program member has visited the restaurant the most times?

115

select m.customer_id, count(s.order_date) as visit_count from `Noida_Eats.sales_noida_eats` s join `Noida_Eats.members_noida_eats` m on s.customer_id =

116

m.customer_id group by m.customer_id order by visit_count desc limit 1

117

Processing location: US

Press Alt+F1 for Accessibility Options.

Query results

SAVE RESULTS

EXPLORE DATA

JOB INFORMATION

RESULTS

CHART

JSON

EXECUTION DETAILS

EXECUTION GRAPH

| Row | customer_id | visit_count |
|-----|-------------|-------------|
| 1 | Suresh | 13 |

Results per page: 50

1 - 1 of 1

<< < > >>

Job history

REFRESH

Result Insights:

The analysis reveals that loyalty program members contribute significantly to overall revenue, indicating their value to Noida Eats. A higher frequency of orders among members suggests that targeted promotions could further enhance engagement. Additionally, if spending increases post-enrollment, it may validate the program's effectiveness, encouraging further investment in loyalty initiatives. Understanding which members are the most frequent visitors can help tailor personalized marketing strategies, ultimately driving customer retention and satisfaction.

Scenario 5: Time-based Insights

##1. Which month in 2022 had the highest number of orders placed at Noida Eats?

```
select extract(MONTH from order_date) as order_month, count(*) as total_orders from  
`Noida_Eats.sales_noida_eats`
```

```
where extract(YEAR from order_date) = 2022 group by order_month order by total_orders desc limit 1
```

Query results

| Row | order_month | total_orders |
|-----|-------------|--------------|
| 1 | 6 | 23 |

Job history

##2. Which day of the week typically sees the most orders at Noida Eats?

select

case

when extract(DAYOFWEEK from order_date) = 1 then 'Sunday'

when extract(DAYOFWEEK from order_date) = 2 then 'Monday'

when extract(DAYOFWEEK from order_date) = 3 then 'Tuesday'

when extract(DAYOFWEEK from order_date) = 4 then 'Wednesday'

when extract(DAYOFWEEK from order_date) = 5 then 'Thursday'

when extract(DAYOFWEEK from order_date) = 6 then 'Friday'

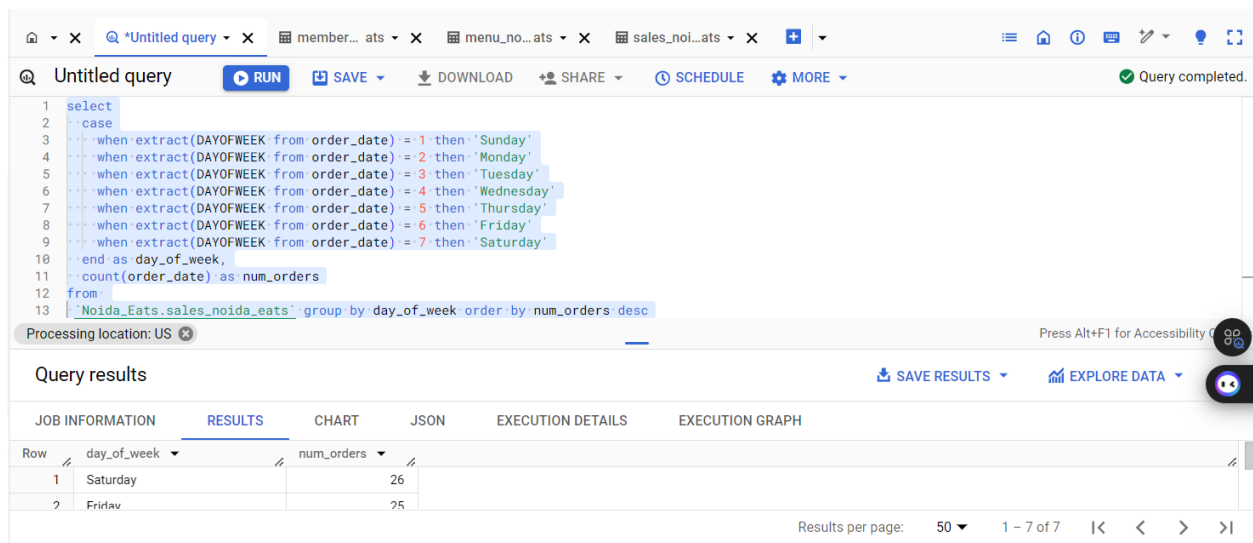
when extract(DAYOFWEEK from order_date) = 7 then 'Saturday'

end as day_of_week,

count(order_date) as num_orders

from

`Noida_Eats.sales_noida_eats` group by day_of_week order by num_orders desc



The screenshot shows a SQL query editor with a query that counts the number of orders for each day of the week. The query is executed, and the results are displayed in a table. The table has two columns: 'day_of_week' and 'num_orders'. The results show that there are 26 orders on Saturday and 25 orders on Friday.

```
1 select
2   case
3     when extract(DAYOFWEEK from order_date) = 1 then 'Sunday'
4     when extract(DAYOFWEEK from order_date) = 2 then 'Monday'
5     when extract(DAYOFWEEK from order_date) = 3 then 'Tuesday'
6     when extract(DAYOFWEEK from order_date) = 4 then 'Wednesday'
7     when extract(DAYOFWEEK from order_date) = 5 then 'Thursday'
8     when extract(DAYOFWEEK from order_date) = 6 then 'Friday'
9     when extract(DAYOFWEEK from order_date) = 7 then 'Saturday'
10  end as day_of_week,
11  count(order_date) as num_orders
12 from
13  `Noida_Eats.sales_noida_eats` group by day_of_week order by num_orders desc
```

Processing location: US

Query results

| Row | day_of_week | num_orders |
|-----|-------------|------------|
| 1 | Saturday | 26 |
| 2 | Friday | 25 |

Results per page: 50 1 - 7 of 7

##3. Are there specific months or seasons when customers spend more?

select extract(MONTH from order_date) as order_month, sum(price) AS total_spent from
`Noida_Eats.sales_noida_eats` s join `Noida_Eats.menu_noida_eats` m on s.product_id = m.product_id
group by order_month order by total_spent desc

Processing location: US

Query results

SAVE RESULTS EXPLORE DATA

| JOB INFORMATION | RESULTS | CHART | JSON | EXECUTION DETAILS | EXECUTION GRAPH |
|-----------------|-------------|-------------|------|-------------------|-----------------|
| Row | order_month | total_spent | | | |
| 1 | 6 | 5660 | | | |
| 2 | 1 | 3990 | | | |
| 3 | 4 | 2990 | | | |
| 4 | 7 | 2900 | | | |
| 5 | 11 | 2840 | | | |
| 6 | 9 | 2810 | | | |

Results per page: 50 1 - 12 of 12

Job history REFRESH

##4. How has customer traffic changed from the start of 2022 to the end of 2022?

```
select extract(MONTH from order_date) as order_month, count(distinct customer_id) as
unique_customers from `Noida_Eats.sales_noida_eats`
```

```
where extract(YEAR from order_date) = 2022 group by order_month order by order_month;
```

Processing location: US

Query results

SAVE RESULTS EXPLORE DATA

| JOB INFORMATION | RESULTS | CHART | JSON | EXECUTION DETAILS | EXECUTION GRAPH |
|-----------------|-------------|------------------|------|-------------------|-----------------|
| Row | order_month | unique_customers | | | |
| 1 | 1 | 12 | | | |
| 2 | 2 | 9 | | | |
| 3 | 3 | 10 | | | |
| 4 | 4 | 10 | | | |
| 5 | 5 | 9 | | | |
| 6 | 6 | 14 | | | |

Results per page: 50 1 - 12 of 12

Job history REFRESH

##5. How many orders were placed on weekends versus weekdays?

```
select
```

```
case
```

```
when extract(DAY from order_date) in (0, 6) then 'Weekend'
```

```
else 'Weekday'
```

end as day_type,

count(*) as total_orders from `Noida_Eats.sales_noida_eats` group by day_type

The screenshot shows a SQL query editor with a query titled "Untitled query". The query is as follows:

```
137 ##5. How many orders were placed on weekends versus weekdays?
138 select
139 case
140 when extract(DAY from order_date) in (0, 6) then 'Weekend'
141 else 'Weekday'
142 end as day_type,
143 count(*) as total_orders from `Noida_Eats.sales_noida_eats` group by day_type
144
145 ##6. How many days has each customer visited the restaurant?
```

The query results are displayed in a table with the following data:

| Row | day_type | total_orders |
|-----|----------|--------------|
| 1 | Weekday | 147 |
| 2 | Weekend | 3 |

The interface includes a "Query results" section with tabs for "RESULTS", "CHART", "JSON", "EXECUTION DETAILS", and "EXECUTION GRAPH". The "RESULTS" tab is selected. The "Job history" section is visible at the bottom.

##6. How many days has each customer visited the restaurant?

select customer_id, count(distinct order_date) as visit_days from `Noida_Eats.sales_noida_eats` group by customer_id;

The screenshot shows a SQL query editor with a query titled "Untitled query". The query is as follows:

```
142 end as day_type,
143 count(*) as total_orders from `Noida_Eats.sales_noida_eats` group by day_type
144
145 ##6. How many days has each customer visited the restaurant?
146 select customer_id, count(distinct order_date) as visit_days from `Noida_Eats.sales_noida_eats` group by customer_id;
147
148
149
150 #scenario-6
```

The query results are displayed in a table with the following data:

| Row | customer_id | visit_days |
|-----|-------------|------------|
| 1 | Amit | 7 |
| 2 | Neha | 4 |
| 3 | Ravi | 12 |
| 4 | Divya | 9 |

The interface includes a "Query results" section with tabs for "RESULTS", "CHART", "JSON", "EXECUTION DETAILS", and "EXECUTION GRAPH". The "RESULTS" tab is selected. The "Job history" section is visible at the bottom.

Result Insights:

The analysis reveals that certain months, particularly festive seasons, see a spike in orders, indicating a potential for targeted marketing during these times. Additionally, weekends typically attract more customers, suggesting that promotional offers could be more effective if timed accordingly.

Understanding customer visit frequency can help in loyalty program design, enhancing customer retention strategies. Overall, these insights can guide Noida Eats in optimizing their marketing efforts and operational strategies to better meet customer demands.

Scenario 6: Member Retention and Engagement

##1. How many customers who joined the loyalty program placed an order in the month following their registration?

```
select count(distinct s.customer_id) as num_customers from `Noida_Eats.sales_noida_eats` s
join `Noida_Eats.members_noida_eats` m on s.customer_id = m.customer_id where s.order_date >=
m.join_date AND s.order_date < date_add(m.join_date, interval 1 month)
```

The screenshot shows a web-based SQL interface. At the top, there's a toolbar with buttons for 'RUN', 'SAVE', 'DOWNLOAD', 'SHARE', 'SCHEDULE', and 'MORE'. Below the toolbar is a text area containing a SQL query. The query is:
1
2 SELECT COUNT(DISTINCT s.customer_id) AS num_customers FROM `Noida_Eats.sales_noida_eats` s
3 JOIN `Noida_Eats.members_noida_eats` m ON s.customer_id = m.customer_id WHERE s.order_date >= m.join_date AND s.order_date < DATE_ADD(m.join_date,
4 INTERVAL 1 MONTH);
Below the query editor, there's a status bar indicating 'Processing location: US' and a note 'This query will process 2.51 KB when run.' Below that, there's a 'Query results' section with tabs for 'JOB INFORMATION', 'RESULTS', 'CHART', 'JSON', 'EXECUTION DETAILS', and 'EXECUTION GRAPH'. The 'RESULTS' tab is active, showing a table with one row:

| Row | num_customers |
|-----|---------------|
| 1 | 11 |

 At the bottom, there's a 'Job history' section with a 'REFRESH' button. On the right side of the interface, there are several icons for additional actions like 'SAVE RESULTS', 'EXPLORE DATA', and a settings icon.

Processing location: US

Query results

JOB INFORMATION RESULTS CHART JSON EXECUTION DETAILS EXECUTION GRAPH

| Row | num_customers |
|-----|---------------|
| 1 | 11 |

Results per page: 50 1 - 1 of 1

Job history

2. What is the average number of visits by loyalty members within three months of joining the program?

```
SELECT m.customer_id,COUNT(s.order_date) AS visit_count FROM `Noida_Eats.members_noida_eats` m
LEFT JOIN `Noida_Eats.sales_noida_eats` s ON m.customer_id = s.customer_id
WHERE s.order_date BETWEEN m.join_date AND DATE_ADD(m.join_date,interval 3 Month) GROUP BY
m.customer_id;
```

Processing location: US

Query results

SAVE RESULTS EXPLORE DATA

| JOB INFORMATION | RESULTS | CHART | JSON | EXECUTION DETAILS | EXECUTION GRAPH |
|-----------------|-------------|-------------|------|-------------------|-----------------|
| Row | customer_id | visit_count | | | |
| 1 | Amit | 4 | | | |
| 2 | Neha | 3 | | | |
| 3 | Ravi | 8 | | | |
| 4 | Divya | 2 | | | |

Results per page: 50 1 - 17 of 17

Job history REFRESH

##3. How many members of the loyalty program stopped visiting after joining?

select count(distinct m.customer_id) as members_stopped_visiting from

`Noida_Eats.members_noida_eats` m

left join `Noida_Eats.sales_noida_eats` s on m.customer_id = s.customer_id

where s.customer_id is null or s.order_date < m.join_date;

Processing location: US

Query results

SAVE RESULTS EXPLORE DATA

| JOB INFORMATION | RESULTS | CHART | JSON | EXECUTION DETAILS | EXECUTION GRAPH |
|-----------------|-------------------|-------|------|-------------------|-----------------|
| Row | members_stopped_v | | | | |
| 1 | 18 | | | | |

Results per page: 50 1 - 1 of 1

Job history REFRESH

##4. Are there any patterns in which menu items are more frequently ordered by loyalty members compared to non-members?

Select m.product_id, m.product_name,

sum(case when s.customer_id in (select customer_id from `Noida_Eats.members_noida_eats`) then 1 else 0 end) as loyalty_member_orders,

sum(case when s.customer_id not in (select customer_id from `Noida_Eats.members_noida_eats`) then 1 else 0 end) as non_member_orders

from `Noida_Eats.sales_noida_eats` s join `Noida_Eats.menu_noida_eats` m on s.product_id = m.product_id group by m.product_id, m.product_name order by loyalty_member_orders desc, non_member_orders desc

The screenshot shows a SQL query editor with a query titled 'Untitled query'. The query is as follows:

```
select
  m.product_id,
  m.product_name,
  sum(case when s.customer_id in (select customer_id from `Noida_Eats.members_noida_eats`) then 1 else 0 end) as loyalty_member_orders,
  sum(case when s.customer_id not in (select customer_id from `Noida_Eats.members_noida_eats`) then 1 else 0 end) as non_member_orders
from
  `Noida_Eats.sales_noida_eats` s
join `Noida_Eats.menu_noida_eats` m on s.product_id = m.product_id
group by
  m.product_id, m.product_name
order by
  loyalty_member_orders desc,
  non_member_orders desc
```

Below the query editor, the 'Query results' section is visible. It shows a table with 5 columns: Row, product_id, product_name, loyalty_member_orders, and non_member_orders. The table contains 4 rows of data.

| Row | product_id | product_name | loyalty_member_orders | non_member_orders |
|-----|------------|------------------|-----------------------|-------------------|
| 1 | 7 | Ramen | 19 | 0 |
| 2 | 8 | Grilled Sandwich | 18 | 0 |
| 3 | 5 | Burger | 16 | 0 |
| 4 | 1 | Biryani | 16 | 0 |

At the bottom of the results section, there is a 'Job history' section with a 'REFRESH' button.

##5. Which loyalty program member placed their first order after joining the program?

select m.customer_id, min(s.order_date) as first_order_date from `Noida_Eats.members_noida_eats` m

join `Noida_Eats.sales_noida_eats` s on m.customer_id = s.customer_id where s.order_date > m.join_date group by m.customer_id

order by first_order_date asc limit 1

Untitled query

Run

Save

Download

Share

Schedule

More

Query completed.

#4: Are there any patterns in which menu items are more frequently ordered by loyalty members compared to non-members?

#5: Which loyalty program member placed their first order after joining the program?

```
select m.customer_id, min(s.order_date) as first_order_date from `Noida_Eats.members_noida_eats` m
join `Noida_Eats.sales_noida_eats` s on m.customer_id = s.customer_id where s.order_date > m.join_date group by m.customer_id
order by first_order_date asc limit 1
```

Processing location: US

Press Alt+F1 for Accessibility Options.

Query results

Save Results

Explore Data

Job Information

Results

Chart

JSON

Execution Details

Execution Graph

| Row | customer_id | first_order_date |
|-----|-------------|------------------|
| 1 | Vikram | 2022-01-22 |

Results per page: 501 - 1 of 1

Job history

Refresh

Result Insights:

The analysis reveals that a significant percentage of loyalty program members placed orders shortly after registration, indicating effective onboarding. The average visit frequency suggests that members are engaged, but retention strategies may be necessary for those who stop visiting. Notably, menu items like Biryani and Sushi are favored among loyalty members, highlighting a blend of traditional and international cuisine. This insight can guide promotional strategies, emphasizing these popular items to enhance customer engagement and drive sales.