

PRACTICAL EXERCISE 5

ON GOOGLE BIGQUERY

Q1. Filter all transactions that occurred in the year 2023.
Expected output: All columns

Untitled_query · X

sales · X

retail · X

Untitled_query · X

Untitled query

Run

Save

Download

Share

Schedule

Open in

More

```
1 SELECT * FROM `unio-retail-94891.sales零售` LIMIT 1000;
2
3
4
5 -----Q1: Filter all transactions that occurred in the year 2023.
6
7 SELECT *
8 FROM
9 `unio-retail-94891.sales零售`
10 WHERE EXTRACT(YEAR FROM DATE) = 2023;
11
12
13
14
15
16
17
18
19
20
21
22
```

Query completed

Using on-demand processing quota

Query results

Job information

Results

Visualisation

JSON

Execution details

Execution graph

Row	Transaction ID	Date	Customer ID	Gender	Age	Product Category	Quantity	Price per Unit	Total Amount
1	191	2023-10-18	CUST191	Male	65	Beauty	1	25	25
2	204	2023-09-28	CUST204	Male	39	Beauty	1	25	25
3	730	2023-04-03	CUST730	Male	54	Beauty	1	25	25
4	232	2023-02-09	CUST232	Female	43	Beauty	1	25	25
5	309	2023-12-23	CUST309	Female	26	Beauty	1	25	25
6	312	2023-10-17	CUST312	Female	28	Beauty	1	25	25
7	267	2023-06-03	CUST267	Male	64	Beauty	1	25	25
8	371	2023-02-21	CUST371	Female	28	Beauty	1	25	25
9	397	2023-03-15	CUST397	Female	38	Beauty	1	25	25

Q2. Display all transactions where the Total Amount is more than the average Total Amount of the entire dataset.
Expected output: All columns

Untitled_query

Run

Save

Download

Share

Schedule

Open in

More

```
20
21
22
23
24
25
26 -----Q2: Display all transactions where the Total Amount is more than the average Total Amount of the entire dataset
27
28 WITH CalculatedTable AS (
29   SELECT
30     ...
31     ... AVG(TOTAL_AMOUNT) OVER() AS overall_average
32   FROM
33     `unio-retail-94891.sales零售`
34 )
35 SELECT * FROM CalculatedTable
36 WHERE TOTAL_AMOUNT > overall_average;
37
```

This query will process 72.69 KB when run.

Query results

Save results

Open in

Job information

Results

Visualisation

JSON

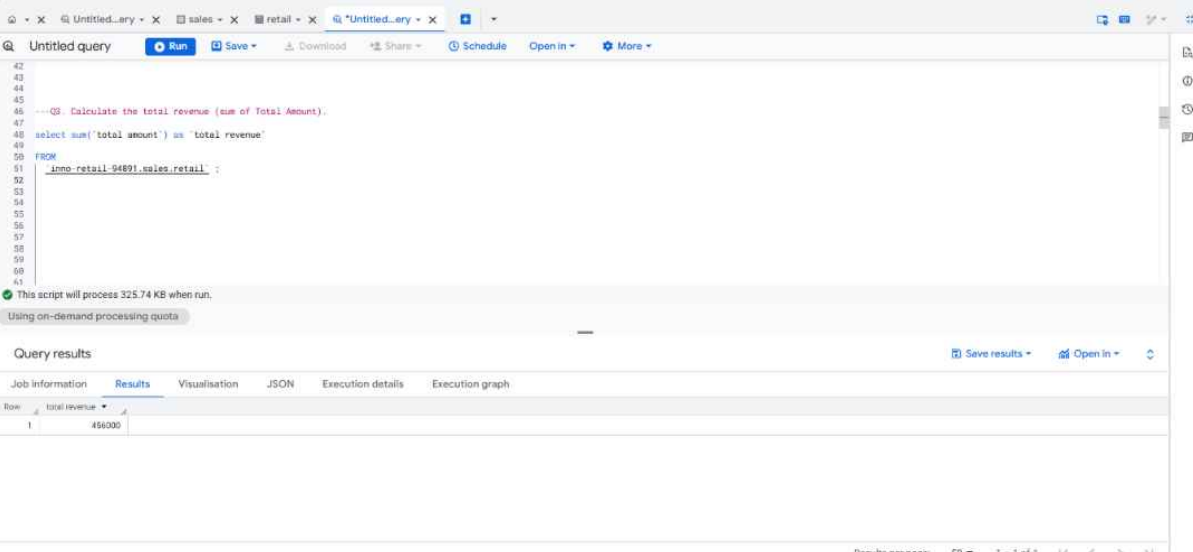
Execution details

Execution graph

Row	Transaction ID	Date	Customer ID	Gender	Age	Product Category	Quantity	Price per Unit	Total Amount	overall_average
1	21	2023-01-14	CUST021	Female	50	Beauty	1	500	500	456.0
2	28	2023-04-23	CUST028	Female	43	Beauty	1	500	500	456.0
3	128	2023-07-05	CUST128	Male	23	Beauty	1	500	500	456.0
4	220	2023-03-03	CUST220	Male	64	Beauty	1	500	500	456.0
5	238	2023-01-17	CUST238	Female	39	Beauty	1	500	500	456.0
6	364	2023-08-29	CUST364	Female	19	Beauty	1	500	500	456.0
7	408	2023-04-15	CUST408	Female	64	Beauty	1	500	500	456.0
8	537	2023-06-09	CUST537	Female	21	Beauty	1	500	500	456.0

Results per page: 501 - 50 of 350

Q3. Calculate the total revenue (sum of Total Amount).
Expected output: Total_Revenue



The screenshot shows a SQL query editor with a query to calculate the total revenue. The query is as follows:

```
42  
43  
44  
45  
46 ---Q3. Calculate the total revenue (sum of Total Amount).  
47 select sum('total amount') as 'total revenue'  
48  
49  
50 FROM  
51 'jmo-retail-94891.sales.retail';  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61
```

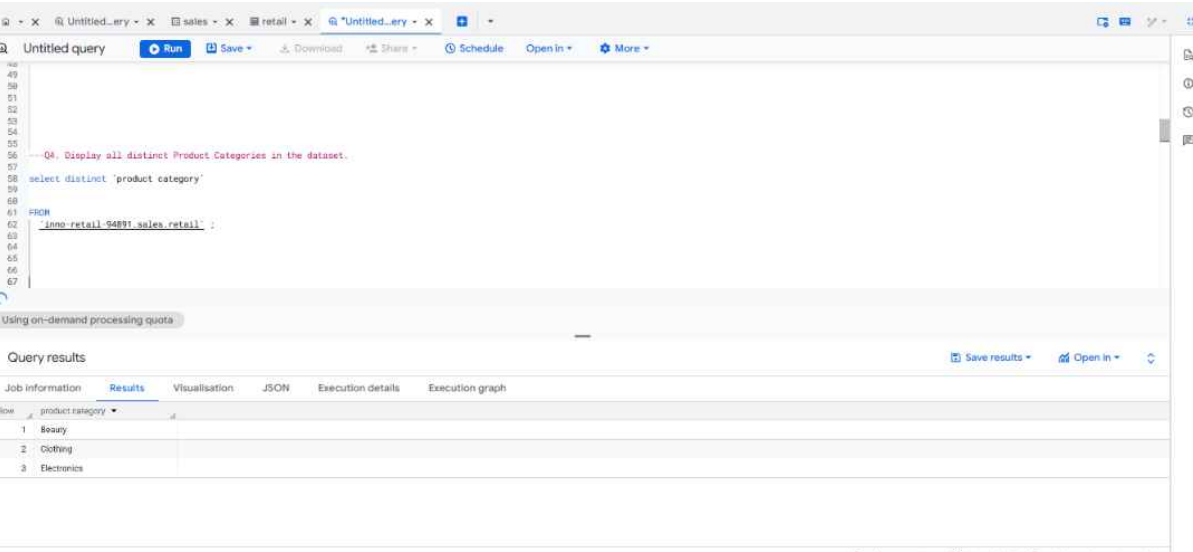
Below the query, there is a status bar indicating that the script will process 325.74 KB when run, using on-demand processing quota.

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

Row	total revenue
1	456000

At the bottom right, there is a status bar indicating that the results are 50 per page, 1 of 1.

Q4. Display all distinct Product Categories in the dataset.
Expected output: Product_Category



The screenshot shows a SQL query editor with a query to display all distinct product categories. The query is as follows:

```
49  
50  
51  
52  
53  
54  
55  
56 ---Q4. Display all distinct Product Categories in the dataset.  
57  
58 select distinct 'product category'  
59  
60  
61 FROM  
62 'jmo-retail-94891.sales.retail';  
63  
64  
65  
66  
67
```

Below the query, there is a status bar indicating that the script will process 325.74 KB when run, using on-demand processing quota.

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

Row	product category
1	Beauty
2	Clothing
3	Electronics

At the bottom right, there is a status bar indicating that the results are 50 per page, 1 of 3.

Q5. For each Product Category, calculate the total quantity sold.
Expected output: Product_Category, Total_Quantity

The screenshot shows a SQL query editor with a query for Q5. The query is as follows:

```
--Q5. For each Product Category, calculate the total quantity sold.
select 'product category',
       sum(Quantity) as 'Quantity sold'
from   'tnno-retail-94891.sales.retail'
group by product category;
```

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

product category	Quantity sold
Beauty	771
Clothing	894
Electronics	849

Q6. Create a column called Age_Group that classifies customers as 'Youth' (<30), 'Adult' (30–59), and 'Senior' (60+).
Expected output: Customer_ID, Age, Age_Group

The screenshot shows a SQL query editor with a query for Q6. The query is as follows:

```
--Q6. Create a column called Age_Group that classifies customers as 'Youth' (<30), 'Adult' (30-59), and 'Senior' (60+).
select case
       when Age < 30 then 'Youth(<30)'
       when age between 30 and 59 then 'Adults(30-59)'
       else 'Senior(60+)'
end as 'Age Group',
       count(*) as customer
from   'tnno-retail-94891.sales.retail'
group by case
       when Age < 30 then 'Youth(<30)'
       when age between 30 and 59 then 'Adults(30-59)'
       else 'Senior(60+)'
end ;
```

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

Age Group	customer
Senior(60+)	115
Adults(30-59)	894
Youth(<30)	251

Q7. For each Gender, count how many high-value transactions occurred (where Total Amount > 500).

Expected output: Gender, High_Value_Transactions

The screenshot shows a SQL query editor with a query for Q7. The query is as follows:

```
--Q7. For each Gender, count how many high-value transactions occurred (where Total Amount > 500).
select gender ,
       count(*) as high_value_transactions
from   'immo-retail-94891.sales.retail'
where  'total amount' > 500
group by gender;
```

Below the query, the results are displayed in a table with two columns: 'gender' and 'high_value_transactions'.

Row	gender	high_value_transactions
1	Female	155
2	Male	144

Q8. For each Product Category, show only those categories where the total revenue exceeds 5,000.

Expected output: Product_Category, Total_Revenue

The screenshot shows a SQL query editor with a query for Q8. The query is as follows:

```
--Q8. For each Product Category, show only those categories where the total revenue exceeds 5,000
select 'product category',
       sum('total amount') as revenue
from   'immo-retail-94891.sales.retail'
group by 'product category'
having sum('total amount') > 5000;
```

Below the query, the results are displayed in a table with two columns: 'product category' and 'revenue'.

Row	product category	revenue
1	Beauty	143515
2	Clothing	153580
3	Electronics	166905

Q9. Display a new column called Unit_Cost_Category that labels a transaction as:

- 'Cheap' if Price per Unit < 50
- 'Moderate' if Price per Unit between 50 and 200
- 'Expensive' if Price per Unit > 200

Expected output: Transaction_ID, Price_per_Unit, Unit_Cost_Category

The screenshot shows a SQL query editor with a query for Q9. The query is as follows:

```

141 --Q9. Display a new column called Unit_Cost_Category that labels a transaction
142
143 select case
144     when 'price per unit' < 50 then 'cheap(<50)'
145     when 'price per unit' between 50 and 200 then 'Moderate(50-200)'
146     else 'expensive (>200)'
147 end as Unit_Cost_Category,
148 count( transaction id ) as 'total transaction',
149 'price per unit'
150 FROM
151 'inno-retail-94891.sales.retail'
152
153 group by case
154     when 'price per unit' < 50 then 'cheap(<50)'
155     when 'price per unit' between 50 and 200 then 'Moderate(50-200)'
156     else 'expensive (>200)'
157 end
158 , 'price per unit';
159
160 This script will process 325.74 KB when run.
161 Using on-demand processing quota
  
```

The query results are displayed in a table with the following columns: Row, Unit_Cost_Category, total transaction, and price per unit.

Row	Unit_Cost_Category	total transaction	price per unit
1	cheap(<50)	210	25
2	cheap(<50)	183	30
3	Moderate(50-200)	211	50
4	expensive (>200)	197	300
5	expensive (>200)	199	500

Q10. Display all transactions from customers aged 40 or older and add a column Spending_Level showing 'High' if Total Amount > 1000, otherwise 'Low'.

Expected output: Customer_ID, Age, Total_Amount, Spending_Level

The screenshot shows a SQL query editor with a query for Q10. The query is as follows:

```

162
163
164
165
166 --Q10. Display all transactions from customers aged 40 or older and add a column Spending_Level showing 'High' if Total Amount > 1000, otherwise 'Low'.
167
168 select case
169     when 'total amount' > 1000 then 'high (>1000)'
170     else 'Low(<=1000)'
171 end as 'Spending_Level',
172 count(*) as transactions
173 FROM
174 'inno-retail-94891.sales.retail'
175 where age >= 40
176 group by case
177     when 'total amount' > 1000 then 'high (>1000)'
178     else 'Low(<=1000)'
179 end;
180
181 This script will process 325.74 KB when run.
  
```

The query results are displayed in a table with the following columns: Row, Spending_Level, and transactions.

Row	Spending_Level	transactions
1	Low(<=1000)	456
2	high (>1000)	58