



E-COMMERCE

PROJECT USING
SQL



I AM KAMAL AND IN THIS
PROJECT I UTILIZED SQL QUERIES
TO SOLVE THE SALES BASED
QUESTIONS.



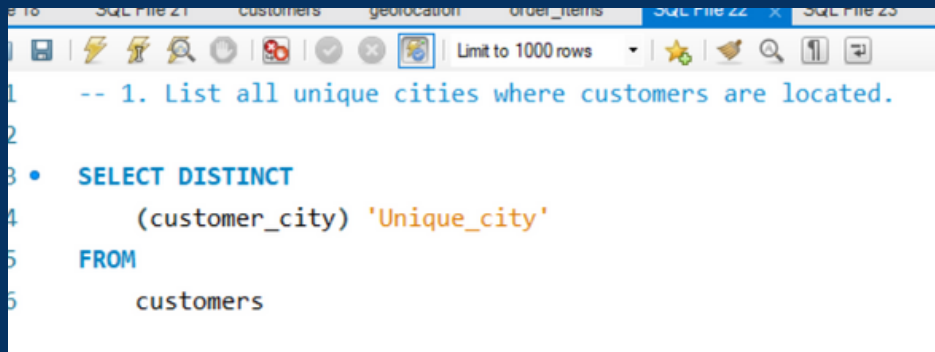
Questions

- List all unique cities where customers are located.
- Count the number of orders placed in 2017.
- Find the total sales per category.
- Calculate the percentage of orders that were paid in installments.
- Count the number of customers from each state.

Questions

- Calculate the number of orders per month in 2018.
- Find the average number of products per order, grouped by customer city.
- Calculate the percentage of total revenue contributed by each product category.
- Identify the correlation between product price and the number of times a product has been purchased.
- Calculate the total revenue generated by each seller, and rank them by revenue.

- List all unique cities where customers are located.



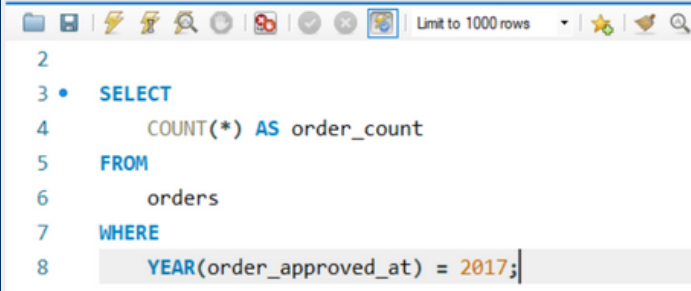
The screenshot shows a SQL IDE window with multiple tabs. The active tab is 'SQL File 22'. The query editor contains the following SQL code:

```
-- 1. List all unique cities where customers are located.  
  
• SELECT DISTINCT  
  (customer_city) 'Unique_city'  
FROM  
  customers
```

output

Unique_city
franca
sao bernardo do campo
sao paulo
mogi das cruze
campinas
jaragua do sul
timoteo
curitiba
belo horizonte



- Count the number of orders placed in 2017.



```
2
3 • SELECT
4     COUNT(*) AS order_count
5 FROM
6     orders
7 WHERE
8     YEAR(order_approved_at) = 2017;
```

The screenshot shows a SQL query editor interface. The query is written in a monospaced font. The editor has a toolbar at the top with various icons for file operations, execution, and search. A dropdown menu on the right indicates 'Limit to 1000 rows'. The query is currently selected, and the text is highlighted in a light gray background.

output

Result Grid			 Filter Rows: <input type="text"/>
	order_count		
▶	44973		

- Find the total sales per category.

```
SQL File 18*  SQL File 21  customers  geolocation  order_items  SQL File 22*  SQL File 23*  SQL File 24* x
Limit to 1000 rows
1  -- Find the total sales per category
2
3  • SELECT pd.product_category, round(sum(p.payment_value)) AS total_sales
4  FROM payments p
5  join order_items ot on ot.order_id = p.order_id
6  join products pd on pd.product_id = ot.product_id
7  group by pd.product_category
8
```

output

Result Grid		Filter Rows:	Export:	Wrap
product_category	total_sales			
perfumery	506739			
Furniture Decoration	1430176			
telephony	486882			
bed table bath	1712554			
automotive	852294			
computer accessories	1585330			
housewares	1094758			
babies	539846			
toys	619038			
Furniture office	646826			

- Calculate the percentage of orders that were paid in installments.

```
3 • SELECT
4     (COUNT(CASE
5         WHEN payment_type = 'Installments' THEN 1
6     END) * 100.0 / COUNT(*)) AS installment_percentage
7 FROM
8     payments;
```

output

Result Grid		Filter Rows:
	installment_percentage	
▶	0.00000	

- Count the number of customers from each state.

```
1  -- Count the number of customers from each state.
2
3 • select customer_state ,
4     |count(customer_id) 'total customer'|from customers
5     group by customer_state
```

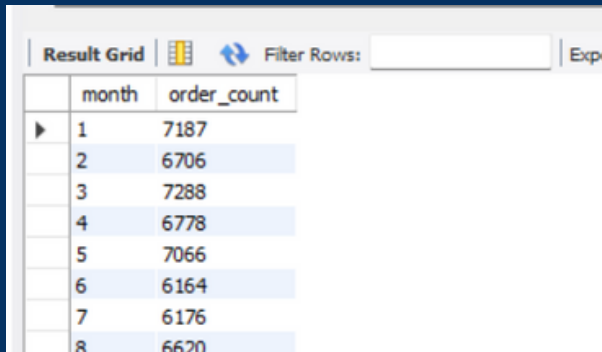
output

Result Grid		Filter Rows:	Exports:	Wrap Cell Cont
customer_state	total customer			
CE	1336			
DF	2140			
RN	485			
PE	1652			
MT	907			
AM	148			
AP	68			
AL	413			
RO	253			
PB	536			

- Calculate the number of orders per month in 2018.

```
1  -- Calculate the number of orders per month in 2018.
2
3  • SELECT MONTH(order_approved_at) AS month,
4     COUNT(*) AS order_count
5  FROM orders
6  WHERE YEAR(order_approved_at) = 2018
7  GROUP BY MONTH(order_approved_at)
8  ORDER BY month;
```

output



The screenshot shows a database interface with a 'Result Grid' tab. It displays the output of the SQL query, showing the month and the corresponding order count for each month in 2018. The table has two columns: 'month' and 'order_count'. The data is as follows:

month	order_count
1	7187
2	6706
3	7288
4	6778
5	7066
6	6164
7	6176
8	6620

- Find the average number of products per order, grouped by customer city.

```
1  -- Find the average number of
2  -- products per order, grouped by customer city.
3
4  • SELECT customer_city,
5     AVG(od.order_id) AS avg_products_per_order
6  FROM customers c
7  join orders od on od.customer_id = c.customer_id
8  GROUP BY customer_city;
```

output

customer_city	avg_products_per_order
vianopolis	15948.666666666666
ouro preto	0
goiania	0
feira de santana	3.297297297297297e267
sao paulo	0
cruz das almas	9.987184082568421e306
tocos	8.988465674311579e307
capelinha	2377.9
franca	0
guaruja	0
januaria	8.171332431192344e306

- Calculate the percentage of total revenue contributed by each product category.

```
1  -- Calculate the percentage of total revenue contributed by each product category
2
3  • SELECT pd.product_category,
4      round(SUM(p.payment_value)) AS total_category_revenue,
5      round((SUM(p.payment_value) * 100.0 / total_revenue.total)) AS revenue_percentage
6  FROM payments p
7  JOIN order_items ot ON p.order_id = ot.order_id
8  JOIN products pd ON pd.product_id = ot.product_id
9  CROSS JOIN (SELECT SUM(payment_value) AS total FROM payments) total_revenue
10 GROUP BY pd.product_category, total_revenue.total;
```

output

Result Grid			
Filter Rows:			
Export:			
Wrap Cell Contents			
	product_category	total_category_revenue	revenue_percentage
▶	perfumery	506739	3
	Furniture Decoration	1430176	9
	telephony	486882	3
	bed table bath	1712554	11
	automotive	852294	5
	computer accessories	1585330	10
	housewares	1094758	7
	babies	539846	3
	toys	619038	4
	Furniture accessories	646976	4

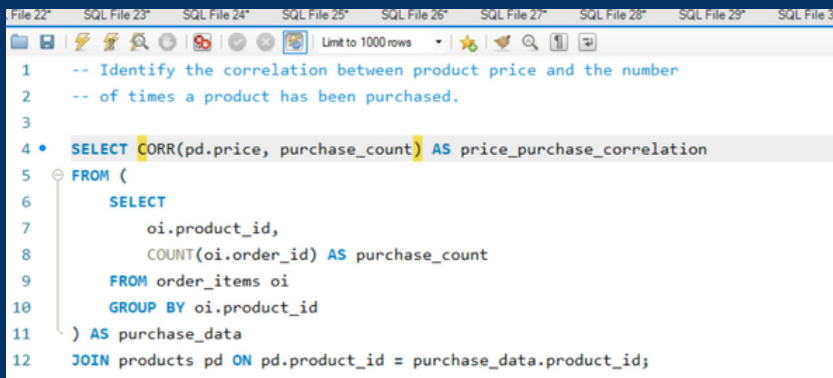
- Identify the correlation between product price and the number of times a product has been purchased.

```
1  -- Calculate the total revenue generated by each seller, and rank them by revenue
2
3  • SELECT
4      s.seller_id,
5      SUM(p.payment_value) AS total_revenue,
6      RANK() OVER (ORDER BY SUM(p.payment_value) DESC) AS revenue_rank
7  FROM payments p
8  join order_items ot on ot.order_id = p.order_id
9  join sellers s on s.seller_id = ot.seller_id
10 GROUP BY s.seller_id
11 ORDER BY total_revenue DESC;
```

output

Result Grid				Filter Rows:	Export:	Wrap Cell Contents:
	seller_id	total_revenue	revenue_rank			
▶	7c67e1448b00f6e969d365cea6b010ab	507166.9073021412	1			
	1025f0e2d44d7041d6cf58b6550e0bfa	308222.0398402214	2			
	4a3ca9315b744ce9f8e9374361493884	301245.26976528764	3			
	1f50f920176fa81dab994f9023523100	290253.42012761533	4			
	53243585a1d6dc2643021fd1853d8905	284903.0804977417	5			
	da8622b14eb17ae2831f4ac5b9dab84a	272219.31931465864	6			
	4869f7a5dfa277a7dca6462dcf3b52b2	264166.1209387779	7			
	955fee9216a65b617aa5c0531780ce60	236322.30050226487	8			
	fa1c13f2614d7b5c4749dbc52fecda94	206513.22986984253	9			
	7c02a42ef0c4602f628b202420bc752a	185124.2002620646	10			

- Calculate the total revenue generated by each seller, and rank them by revenue.



The screenshot shows a SQL IDE with multiple tabs labeled 'SQL File 22*' through 'SQL File 30*'. The active window displays a SQL query. The query starts with two comments: '-- Identify the correlation between product price and the number' and '-- of times a product has been purchased.' The main query is a SELECT statement with a subquery. The subquery selects 'oi.product_id' and 'COUNT(oi.order_id) AS purchase_count' from 'order_items oi', grouped by 'oi.product_id'. The subquery is aliased as 'purchase_data'. The main query then selects 'CORR(pd.price, purchase_count) AS price_purchase_correlation' from the subquery, joined to 'products pd' on the condition 'pd.product_id = purchase_data.product_id'.

```
1  -- Identify the correlation between product price and the number
2  -- of times a product has been purchased.
3
4  • SELECT CORR(pd.price, purchase_count) AS price_purchase_correlation
5  FROM (
6      SELECT
7          oi.product_id,
8          COUNT(oi.order_id) AS purchase_count
9      FROM order_items oi
10     GROUP BY oi.product_id
11 ) AS purchase_data
12 JOIN products pd ON pd.product_id = purchase_data.product_id;
```