

Beginner Queries.

What is the distribution of order values across all customers in the dataset?

The screenshot shows a SQL query in the Query Editor window, which is set to 'Limit to 1000 rows'. The query is as follows:

```
1  -- To find the distribution of order values across all customers, you can use the following query:
2  • select CustomerId, sum(Quantity * UnitPrice)
3    from onlineretail
4    group by CustomerId;
```

The Results pane displays the output of the query in a grid format. The columns are 'CustomerId' and 'sum(Quantity * UnitPrice)'. The data is as follows:

CustomerId	sum(Quantity * UnitPrice)
13758	362.45000000000005
13694	842.12
15983	440.89
14649	355.83999999999999
17968	277.34999999999997
16210	2474.7399999999993
17897	165.89
17377	223.90000000000003

The Output pane shows the execution of the query, with the following messages:

#	Time	Action	Message	Duration / Fetch
43	16:57:13	SELECT YEAR(str_to_date(InvoiceDate,'m/%d/%y')) AS Year, MONTH(str_to_date(In...	4 row(s) returned	0.094 sec / 0.000 sec
44	17:02:28	SELECT InvoiceNo FROM 'project1'.'onlineretail' LIMIT 0, 1000	1000 row(s) returned	0.000 sec / 0.000 sec

How many unique products has each customer purchased?

The screenshot shows a SQL query in the Query Editor window, which is set to 'Limit to 1000 rows'. The query is as follows:

```
1  -- To count the number of unique products each customer has purchased:
2  • select CustomerId, count(DISTINCT StockCode) as uniqueproduct
3    from onlineretail
4    group by CustomerId;
```

The Results pane displays the output of the query in a grid format. The columns are 'CustomerId' and 'uniqueproduct'. The data is as follows:

CustomerId	uniqueproduct
12431	14
12433	73
12557	5
12583	20
12600	2
12662	15
12682	16
12686	7
12717	0

The Output pane shows the execution of the query, with the following messages:

#	Time	Action	Message	Duration / Fetch
43	16:57:13	SELECT YEAR(str_to_date(InvoiceDate,'m/%d/%y')) AS Year, MONTH(str_to_date(In...	4 row(s) returned	0.094 sec / 0.000 sec
44	17:02:28	SELECT InvoiceNo FROM 'project1'.'onlineretail' LIMIT 0, 1000	1000 row(s) returned	0.000 sec / 0.000 sec

Which customers have only made a single purchase from the company?

The screenshot shows a SQL query in the 'Query Editor' window of SQL Server Enterprise Manager. The query is designed to find customers who have made only a single purchase. The query text is as follows:

```

1  -- To find customers who have made only a single purchase:
2  * select customerId
3  from onlineretail
4  group by CustomerID
5  having count(DISTINCT InvoiceNo)=1;

```

The 'Results' pane shows a table with one column, 'customerId', and 10 rows of data. The 'Output' pane shows the execution of the query, with a message indicating that 1000 rows were returned.

customerId
12395
12427
12431
12433
12557
12583
12600
12662
12663
12664

Which products are most commonly purchased together by customers in the dataset?

The screenshot shows a SQL query in the 'Query Editor' window of SQL Server Enterprise Manager. The query is designed to find products that are most commonly purchased together. The query text is as follows:

```

1  -- To find which products are most commonly purchased together:
2  * SELECT a.StockCode AS Product1, b.StockCode AS Product2, COUNT(*) AS CustomerID
3  FROM onlineretail a
4  JOIN onlineretail b ON a.InvoiceNo = b.InvoiceNo AND a.StockCode < b.StockCode
5  GROUP BY Product1, Product2
6  ORDER BY CustomerID DESC;
7

```

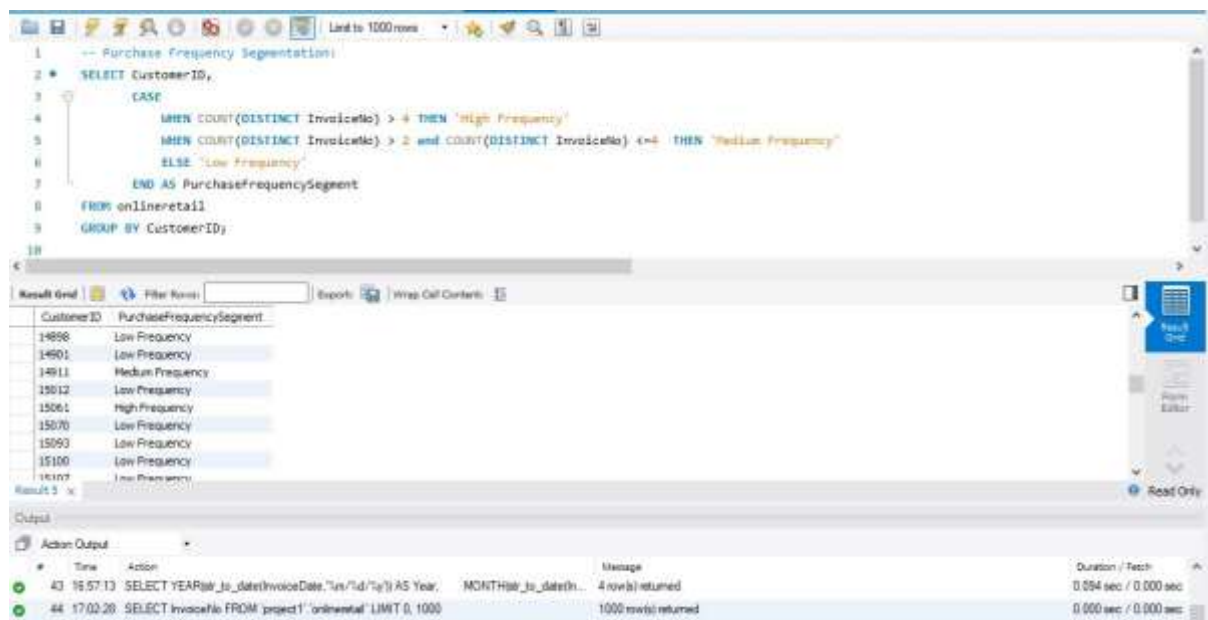
The 'Results' pane shows a table with three columns: 'Product1', 'Product2', and 'CustomerID'. The 'Output' pane shows the execution of the query, with a message indicating that 1000 rows were returned.

Product1	Product2	CustomerID
22632	22633	26
84029E	84029G	22
84029E	85123A	20
84029G	85123A	20
21730	85123A	18
71053	85123A	18
22865	22866	18
22752	85123A	17
71791	71791	17

Advance Queries

1. Customer Segmentation by Purchase Frequency

Group customers into segments based on their purchase frequency, such as high, medium, and low frequency customers. This can help you identify your most loyal customers and those who need more attention.



```
-- Purchase Frequency Segmentation
1
2 * SELECT CustomerID,
3     CASE
4         WHEN COUNT(DISTINCT InvoiceNo) > 4 THEN 'High Frequency'
5         WHEN COUNT(DISTINCT InvoiceNo) > 2 and COUNT(DISTINCT InvoiceNo) <= 4 THEN 'Medium Frequency'
6         ELSE 'Low Frequency'
7     END AS PurchaseFrequencySegment
8 FROM onlineretail
9 GROUP BY CustomerID;
```

CustomerID	PurchaseFrequencySegment
14898	Low Frequency
14901	Low Frequency
14911	Medium Frequency
15012	Low Frequency
15061	High Frequency
15070	Low Frequency
15093	Low Frequency
15100	Low Frequency
16107	Low Frequency

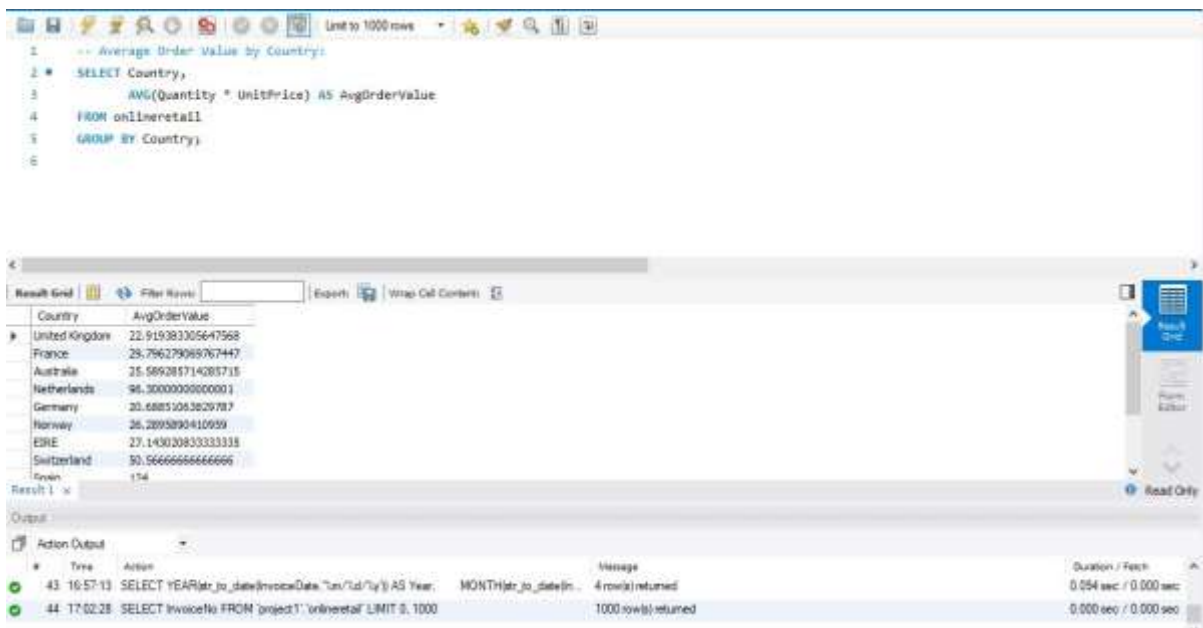
Result 5

Output

#	Time	Action	Message	Duration / Tech
43	16:57:13	SELECT YEAR(is_date)InvoiceDate,"is"/id"/ig") AS Year, MONTH(is_date)...	4 row(s) returned	0.094 sec / 0.000 sec
44	17:02:20	SELECT InvoiceNo FROM 'project1'. 'onlinetail' LIMIT 0, 1000	1000 row(s) returned	0.000 sec / 0.000 sec

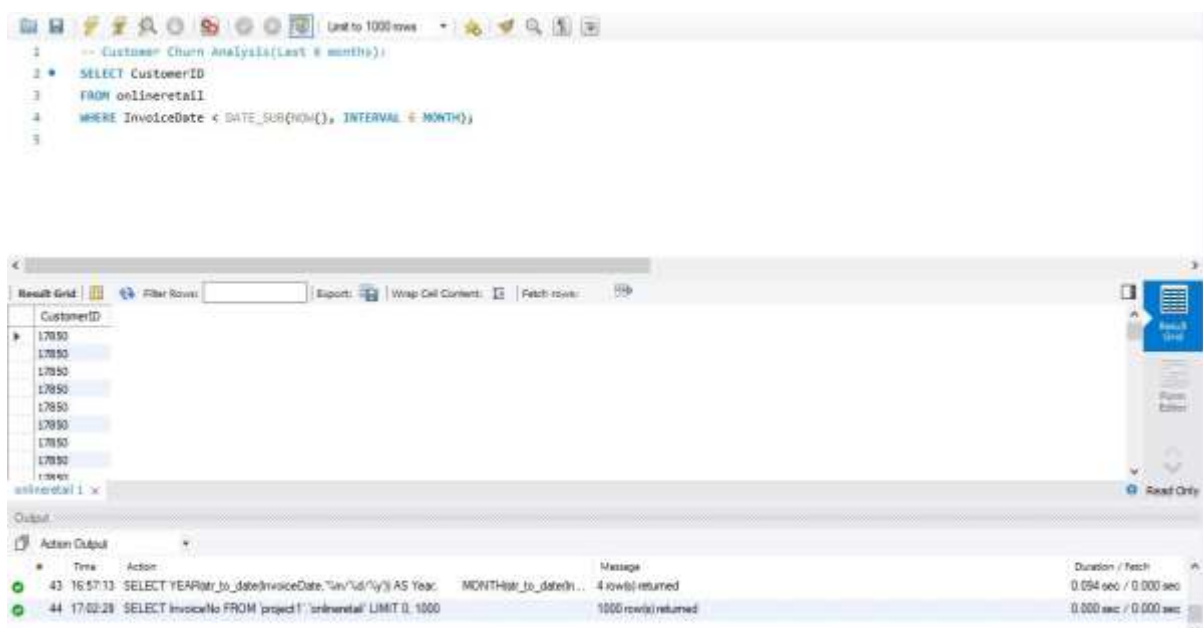
Average Order Value by Country

Calculate the average order value for each country to identify where your most valuable customers are located.



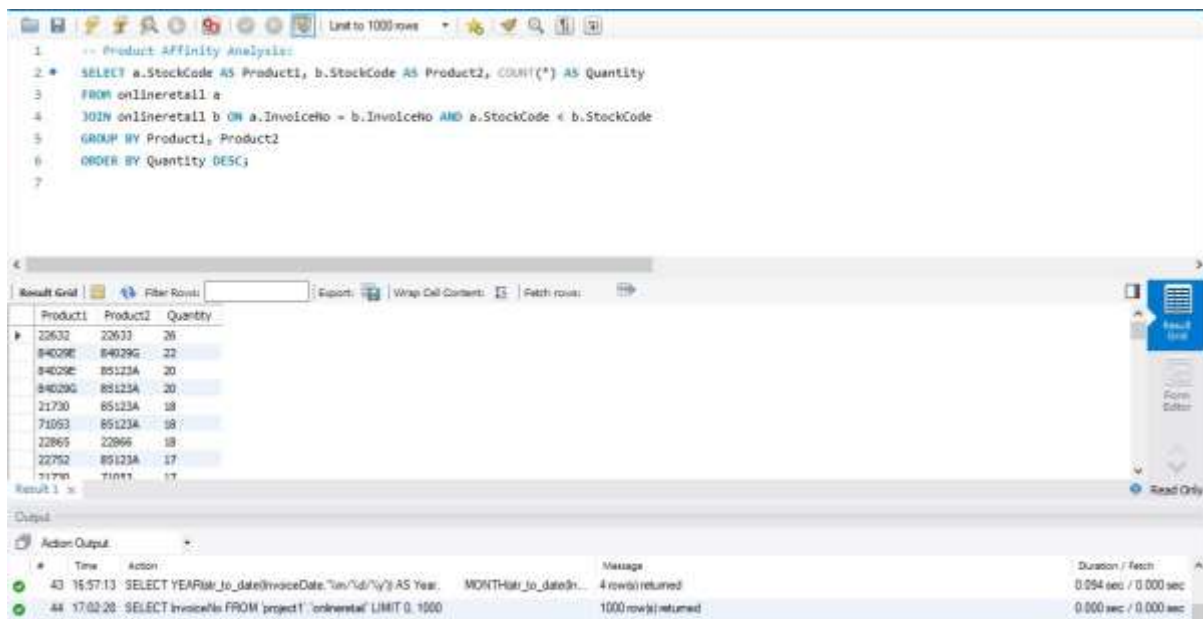
Customer Churn Analysis

Identify customers who haven't made a purchase in a specific period (e.g., last 6 months) to assess churn.



Product Affinity Analysis

Determine which products are often purchased together by calculating the correlation between product purchases.



The screenshot shows a SQL IDE with a query titled "Product Affinity Analysis". The query is as follows:

```
1. -- Product Affinity Analysis
2. * SELECT a.StockCode AS Product1, b.StockCode AS Product2, COUNT(*) AS Quantity
3. FROM onlinetail a
4. JOIN onlinetail b ON a.InvoiceNo = b.InvoiceNo AND a.StockCode < b.StockCode
5. GROUP BY Product1, Product2
6. ORDER BY Quantity DESC;
```

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

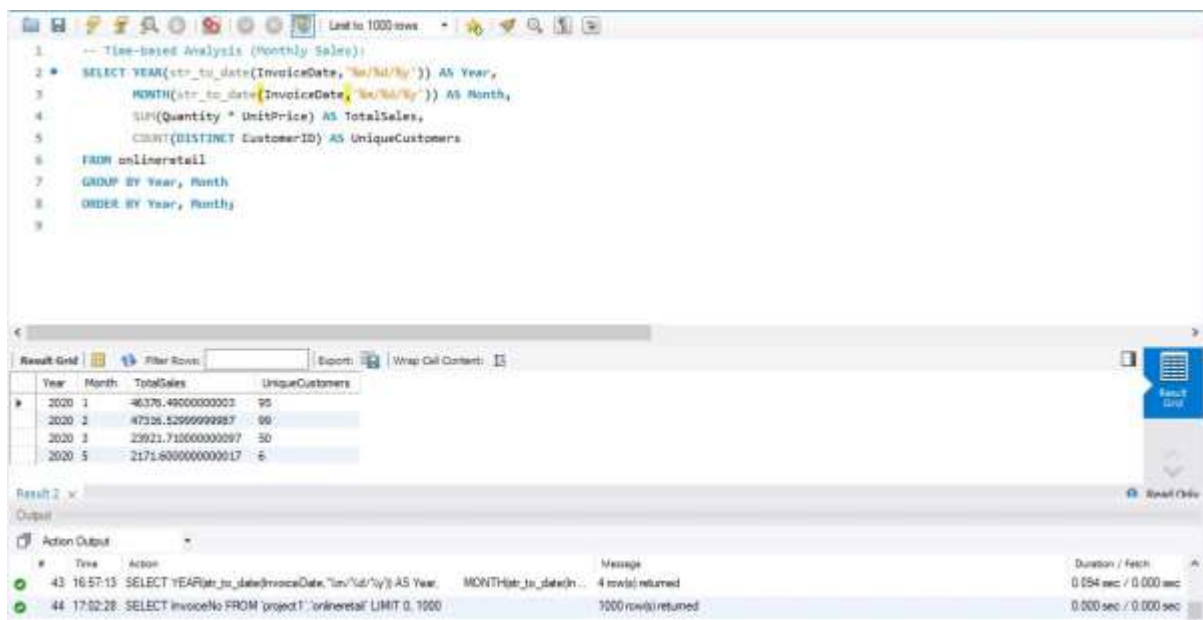
Product1	Product2	Quantity
22632	22633	26
84029E	84029G	22
84029E	85123A	20
84029G	85123A	20
21730	85123A	18
71053	85123A	18
22865	22866	18
22752	85123A	17
71776	71777	17

The Action Output pane shows the following messages:

#	Time	Action	Message	Duration / Fetch
43	16:57:13	SELECT YEAR(str_to_date(InvoiceDate, '%m/%d/%y')) AS Year, MONTH(str_to_date(InvoiceDate, '%m/%d/%y')) AS Month	4 row(s) returned	0.054 sec / 0.000 sec
44	17:02:28	SELECT InvoiceNo FROM 'project1' onlinetail LIMIT 0.1000	1000 row(s) returned	0.000 sec / 0.000 sec

Time-based Analysis

Explore trends in customer behavior over time, such as monthly or quarterly sales patterns.



The screenshot shows a SQL IDE with a query titled "Time-based Analysis (Monthly Sales)". The query is as follows:

```
1. -- Time-based Analysis (Monthly Sales)
2. * SELECT YEAR(str_to_date(InvoiceDate, '%m/%d/%y')) AS Year,
3. MONTH(str_to_date(InvoiceDate, '%m/%d/%y')) AS Month,
4. SUM(Quantity * UnitPrice) AS TotalSales,
5. COUNT(DISTINCT CustomerID) AS UniqueCustomers
6. FROM onlinetail
7. GROUP BY Year, Month
8. ORDER BY Year, Month;
```

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

Year	Month	TotalSales	UniqueCustomers
2020	1	46376.490000000003	95
2020	2	47326.529999999997	96
2020	3	23921.710000000000	50
2020	5	2171.600000000000	6

The Action Output pane shows the following messages:

#	Time	Action	Message	Duration / Fetch
43	16:57:13	SELECT YEAR(str_to_date(InvoiceDate, '%m/%d/%y')) AS Year, MONTH(str_to_date(InvoiceDate, '%m/%d/%y')) AS Month	4 row(s) returned	0.054 sec / 0.000 sec
44	17:02:28	SELECT InvoiceNo FROM 'project1' onlinetail LIMIT 0.1000	1000 row(s) returned	0.000 sec / 0.000 sec

