SQL Advanced Query –

Inventory and Sales Management System
Project business intelligence

Avishay Aknin

Total Sales Revenue by Category:

This query calculates the total sales revenue for each category.

Explanation: This query utilizes joins to relate product sales to categories and calculates the total revenue for each category by multiplying product price by quantity sold.

sql

```sql
SELECT c.CategoryName, SUM(p.ProductPrice * i.Quantity) AS TotalRevenue
FROM Includes i
INNER JOIN Products p ON i.ProductID = p.ProductID
INNER JOIN Category c ON p.CategoryID = c.CategoryID
GROUP BY c.CategoryName;
```

	CategoryName	TotalRevenu
1	Appliances	3000
2	Automotive	150
3	Beauty Products	200
4	Books	15
5	Clothing	70
6	dairy food	44
7	Electronics	2400
8	Furniture	2000
9	Jewelry	100
10	others	120
11	Sporting Goods	30
12	Toys	20

Top Selling Products:

This query identifies the top-selling products based on the total quantity sold.

Explanation: By joining the Includes and Products tables and aggregating the quantity sold for each product, this query ranks the products by total quantity sold and selects the top 5.

sql

```sql
SELECT TOP 5 p.ProductName, SUM(i.Quantity) AS TotalQuantitySold
FROM Includes i
INNER JOIN Products p ON i.ProductID = p.ProductID
GROUP BY p.ProductName
ORDER BY TotalQuantitySold DESC;
```

ProductName	TotalQuantitySold
water	12
Perfume	5
Milk	4
Sofa	4
Laptop	3

Monthly Sales Trend:

This query presents the monthly sales trend over time

Explanation: By aggregating sales revenue by month and year, this query offers a trend analysis of monthly sales, facilitating better understanding of sales patterns over time.

sql

```
SELECT YEAR(TDate) AS Year, MONTH(TDate) AS Month, SUM(p.ProductPrice *
i.Quantity) AS MonthlyRevenue

FROM SalesTransaction t

INNER JOIN Includes i ON t.TID = i.TID

INNER JOIN Products p ON i.ProductID = p.ProductID

GROUP BY YEAR(TDate), MONTH(TDate)

ORDER BY Year, Month;
```

	Year	Month	MonthlyRevenue
1	2021	1	120
2	2021	2	44
3	2022	1	2400
4	2023	1	5585

Average Transaction Value by Store:

This query calculates the average transaction value for each store

Explanation: By joining multiple tables and aggregating transaction values by store, this query helps in assessing the average transaction value per store..

sql

```
SELECT s.StoreID, AVG(p.ProductPrice * i.Quantity) AS AvgTransactionValue

FROM SalesTransaction t

INNER JOIN Includes i ON t.TID = i.TID

INNER JOIN Products p ON i.ProductID = p.ProductID

INNER JOIN Stores s ON t.StoreID = s.StoreID

GROUP BY s.StoreID;
```

	StoreID	AvgTransactionValu
1	S1	86
2	S2	60
3	S3	1275
4	S4	3000
5	S5	1050

Customer Purchase Frequency:

This query analyzes the frequency of customer purchases.

Explanation: Utilizing a left join between Customers and SalesTransaction tables, this query counts the number of transactions for each customer, providing insights into purchase frequency.
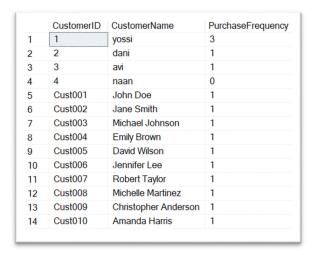
sql

SELECT c.CustomerID, c.CustomerName, COUNT(t.TID) AS PurchaseFrequency

FROM Customers c

LEFT JOIN SalesTransaction t ON c.CustomerID = t.CustomerID

GROUP BY c.CustomerID, c.CustomerName;

	CustomerID	CustomerName	PurchaseFrequency
1	1	yossi	3
2	2	dani	1
3	3	avi	1
4	4	naan	0
5	Cust001	John Doe	1
6	Cust002	Jane Smith	1
7	Cust003	Michael Johnson	1
8	Cust004	Emily Brown	1
9	Cust005	David Wilson	1
10	Cust006	Jennifer Lee	1
11	Cust007	Robert Taylor	1
12	Cust008	Michelle Martinez	1
13	Cust009	Christopher Anderson	1
14	Cust010	Amanda Harris	1

Customer Retention Rate:

This query calculates the customer retention rate by comparing repeat customers over a specified time period.

Explanation: By utilizing a self-join on SalesTransaction table and date calculations, this query computes the retention rate by comparing the count of initial customers with repeat customers over a specified time frame (here, 12 months).

sql

SELECT COUNT(DISTINCT t1.CustomerID) AS InitialCustomers,

COUNT(DISTINCT t2.CustomerID) AS RepeatCustomers,

COUNT(DISTINCT t2.CustomerID) * 100.0 / COUNT(DISTINCT t1.CustomerID) AS RetentionRate

FROM SalesTransaction t1

LEFT JOIN SalesTransaction t2 ON t1.CustomerID = t2.CustomerID

AND DATEDIFF(month, t1.TDate, t2.TDate) > 1

WHERE DATEDIFF(month, t1.TDate, GETDATE()) >= 12;

	InitialCustomers	RepeatCustomers	RetentionRate
1	13	1	7.692307692307

Top Selling Products by Category:

This query ranks products within each category based on total quantity sold.

Explanation: Using the ROW_NUMBER() window function partitioned by category and ordered by quantity sold, this query selects the top-selling product within each category.

sql

```
SELECT CategoryName, ProductName, TotalQuantitySold

FROM)

    SELECT c.CategoryName, p.ProductName, SUM(i.Quantity) AS TotalQuantitySold,

        ROW_NUMBER() OVER (PARTITION BY c.CategoryName ORDER BY
SUM(i.Quantity) DESC) AS Rank

    FROM Includes i

    INNER JOIN Products p ON i.ProductID = p.ProductID

    INNER JOIN Category c ON p.CategoryID = c.CategoryID

    GROUP BY c.CategoryName, p.ProductName

 (AS RankedProducts

WHERE Rank = 1;
```

	CategoryName	ProductName	TotalQuantitySold
1	Appliances	Refrigerator	3
2	Automotive	Car Battery	1
3	Beauty Products	Perfume	5
4	Books	Book: Harry Potter	1
5	Clothing	T-shirt	2
6	dairy food	Milk	4
7	Electronics	Laptop	3
8	Furniture	Sofa	4
9	Jewelry	Necklace	2
10	others	water	12
11	Sporting Goods	Basketball	1
12	Toys	Action Figure	2

Monthly Sales Growth Rate:

This query calculates the monthly sales growth rate compared to the previous month.

Explanation: The query calculates the monthly revenue and adds additional columns to explain the growth rate calculation.

- **'PreviousMonthRevenue'** column fetches the revenue of the previous month using the LAG() function.
- **'RevenueChange'** column calculates the difference between the current month's revenue and the revenue of the previous month.
- **'GrowthRate'** column calculates the growth rate by dividing the revenue change by the revenue of the previous month. This gives the percentage change in revenue from one month to the next.

sql

```sql
SELECT Year ,Month ,MonthlyRevenue,

    LAG(MonthlyRevenue) OVER (ORDER BY Year, Month) AS PreviousMonthRevenue,

    (MonthlyRevenue - LAG(MonthlyRevenue) OVER (ORDER BY Year, Month)) AS
RevenueChange,

    LAG(MonthlyRevenue) OVER (ORDER BY Year, Month) AS PreviousMonthRevenue,

    (MonthlyRevenue - LAG(MonthlyRevenue) OVER (ORDER BY Year, Month)) /
LAG(MonthlyRevenue) OVER (ORDER BY Year, Month) AS GrowthRate

FROM)

    SELECT YEAR(IDate) AS Year ,

        MONTH(IDate) AS Month ,

        SUM(p.ProductPrice * i.Quantity) AS MonthlyRevenue

    FROM SalesTransaction t

    INNER JOIN Includes i ON t.TID = i.TID

    INNER JOIN Products p ON i.ProductID = p.ProductID

    GROUP BY YEAR(IDate), MONTH(IDate)

) AS MonthlySales;
```

	Year	Month	MonthlyRevenue	PreviousMonthRevenue	RevenueChange	PreviousMonthRevenue	GrowthRate
1	2021	1	120	NULL	NULL	NULL	NULL
2	2021	2	44	120	-76	120	0
3	2022	1	2400	44	2356	44	53
4	2023	1	5585	2400	3185	2400	1

Customer Ranking by Total Purchase Amount:

This query ranks customers based on their total purchase amount

Explanation: Utilizing the RANK() window function, this query ranks customers based on their total purchase amount.

sql

```
SELECT CustomerName, TotalPurchaseAmount,

    RANK() OVER (ORDER BY TotalPurchaseAmount DESC) AS CustomerRank

FROM)

    SELECT c.CustomerName, SUM(p.ProductPrice * i.Quantity) AS TotalPurchaseAmount

    FROM Customers c

    INNER JOIN SalesTransaction t ON c.CustomerID = t.CustomerID

    INNER JOIN Includes i ON t.TID = i.TID

    INNER JOIN Products p ON i.ProductID = p.ProductID

    GROUP BY c.CustomerName

  (AS CustomerPurchase;
```

	CustomerName	TotalPurchaseAmount	CustomerRank
1	Robert Taylor	3000	1
2	John Doe	2400	2
3	Emily Brown	2000	3
4	Amanda Harris	200	4
5	Christopher Anderson	150	5
6	dani	120	6
7	Michelle Martinez	100	7
8	Jane Smith	70	8
9	yossi	44	9
10	Jennifer Lee	30	10
11	David Wilson	20	11
12	Michael Johnson	15	12

Running Total Sales Revenue:

This query calculates the running total sales revenue over time.

Explanation: Using the SUM() window function with ORDER BY clause, this query computes the running total of sales revenue over time.

sql

```
SELECT TDate, SUM(TotalRevenue) OVER (ORDER BY TDate) AS RunningTotalRevenue

FROM)

    SELECT TDate, SUM(p.ProductPrice * i.Quantity) AS TotalRevenue

    FROM SalesTransaction t

    INNER JOIN Includes i ON t.TID = i.TID

    INNER JOIN Products p ON i.ProductID = p.ProductID

    GROUP BY TDate

  (AS DailyRevenue;
```

	IDate	RunningTotalRevenue
1	2021-01-11	120
2	2021-02-23	164
3	2022-01-01	2564
4	2023-01-02	2634
5	2023-01-03	2649
6	2023-01-04	4649
7	2023-01-05	4669
8	2023-01-06	4699
9	2023-01-07	7699
10	2023-01-08	7799
11	2023-01-09	7949
12	2023-01-10	8149

Customer Churn Analysis:

This query identifies customers who have not made any purchases in the last 6 months.

Explanation: This query utilizes a nested query to find customers who have not made any purchases in the last 6 months.

sql

```sql
SELECT CustomerID, CustomerName

FROM Customers

WHERE CustomerID NOT IN)

    SELECT DISTINCT t1.CustomerID

    FROM SalesTransaction t1

    WHERE t1.TDate >= DATEADD(month, -6, GETDATE());(
```

	CustomerID	CustomerName
1	1	yossi
2	2	dani
3	3	avi
4	4	naan
5	Cust001	John Doe
6	Cust002	Jane Smith
7	Cust003	Michael Johnson
8	Cust004	Emily Brown
9	Cust005	David Wilson
10	Cust006	Jennifer Lee
11	Cust007	Robert Taylor
12	Cust008	Michelle Martinez
13	Cust009	Christopher Anderson
14	Cust010	Amanda Harris

Average Transaction Value by Month:

This query calculates the average transaction value for each month.

Explanation: Using the AVG() window function partitioned by month and year, this query calculates the average transaction value for each month.

sql

```
SELECT Year, Month, AvgTransactionValue

FROM)

    SELECT YEAR(TDate) AS Year, MONTH(TDate) AS Month,

        AVG(p.ProductPrice * i.Quantity) OVER (PARTITION BY YEAR(TDate),
    MONTH(TDate)) AS AvgTransactionValue

    FROM SalesTransaction t

    INNER JOIN Includes i ON t.TID = i.TID

    INNER JOIN Products p ON i.ProductID = p.ProductID

  (AS MonthlyAvgTransaction;
```

	Year	Month	AvgTransactionValue
1	2021	1	120
2	2021	2	44
3	2022	1	2400
4	2023	1	620
5	2023	1	620
6	2023	1	620
7	2023	1	620
8	2023	1	620
9	2023	1	620
10	2023	1	620
11	2023	1	620
12	2023	1	620

Top 3 Customers by Total Purchase Amount:

This query identifies the top 3 customers based on their total purchase amount.

Explanation: Utilizing the ROW_NUMBER() window function, this query ranks customers based on their total purchase amount and selects the top 3.

sql

```
SELECT CustomerName, TotalPurchaseAmount

FROM)

    SELECT c.CustomerName, SUM(p.ProductPrice * i.Quantity) AS TotalPurchaseAmount,

        ROW_NUMBER() OVER (ORDER BY SUM(p.ProductPrice * i.Quantity) DESC) AS
Rank

    FROM Customers c

    INNER JOIN SalesTransaction t ON c.CustomerID = t.CustomerID

    INNER JOIN Includes i ON t.TID = i.TID

    INNER JOIN Products p ON i.ProductID = p.ProductID

    GROUP BY c.CustomerName

 (AS RankedCustomers

WHERE Rank <= 3;
```

	CustomerName	TotalPurchaseAmount
1	Robert Taylor	3000
2	John Doe	2400
3	Emily Brown	2000

Average Sales Price vs. Global Average:

This query compares the average sales price of products with the global average.

Explanation: Using the AVG() window function, this query compares the average sales price of products with the global average and categorizes them accordingly.

sql

```
SELECT ProductName, ProductPrice ,

    AVG(ProductPrice) OVER () AS GlobalAveragePrice,

    CASE WHEN ProductPrice > AVG(ProductPrice) OVER () THEN 'Above Average'

        WHEN ProductPrice < AVG(ProductPrice) OVER () THEN 'Below Average'

        ELSE 'Equal to Average'

    END AS PriceComparison

FROM Products;
```

	ProductName	ProductPrice	GlobalAveragePrice	PriceComparison
1	Milk	11	161	Below Average
2	water	10	161	Below Average
3	Laptop	800	161	Above Average
4	T-shirt	35	161	Below Average
5	Book: Harry Potter	15	161	Below Average
6	Sofa	500	161	Above Average
7	Action Figure	10	161	Below Average
8	Basketball	30	161	Below Average
9	Refrigerator	1000	161	Above Average
10	Necklace	50	161	Below Average
11	Car Battery	150	161	Below Average
12	Perfume	40	161	Below Average
13	coffe	12	161	Below Average
14	underwear	17	161	Below Average
15	Pants	50	161	Below Average
16	socks	12	161	Below Average
17	milshake	11	161	Below Average

Customer Loyalty Segmentation:

This query segments customers into different loyalty tiers based on their total purchase amount.

Explanation: This query calculates the total purchase amount for each customer and assigns them to different loyalty tiers based on predefined thresholds, allowing for customer segmentation based on loyalty.

sql

```
WITH CustomerPurchase AS)
    SELECT
       c.CustomerName ,
       SUM(p.ProductPrice * i.Quantity) AS TotalPurchaseAmount
    FROM    Customers c
    INNER JOIN
       SalesTransaction t ON c.CustomerID = t.CustomerID
    INNER JOIN
       Includes i ON t.TID = i.TID
    INNER JOIN
       Products p ON i.ProductID = p.ProductID
    GROUP BY
       c.CustomerName
( SELECT    cp.CustomerName ,   cp.TotalPurchaseAmount,
    CASE
       WHEN cp.TotalPurchaseAmount >= 5000 THEN 'Platinum'
       WHEN cp.TotalPurchaseAmount >= 3000 THEN 'Gold'
       WHEN cp.TotalPurchaseAmount >= 1000 THEN 'Silver'
       ELSE 'Bronze'
    END AS LoyaltyTier
FROM   CustomerPurchase cp    ;
```

	CustomerName	TotalPurchaseAmount	LoyaltyTier
1	Amanda Harris	200	Bronze
2	Christopher Anderson	150	Bronze
3	dani	120	Bronze
4	David Wilson	20	Bronze
5	Emily Brown	2000	Silver
6	Jane Smith	70	Bronze
7	Jennifer Lee	30	Bronze
8	John Doe	2400	Silver
9	Michael Johnson	15	Bronze
10	Michelle Martinez	100	Bronze
11	Robert Taylor	3000	Gold
12	yossi	44	Bronze