ONLINE_SALES DATABASE MANAGEMENT SYSTEM

NAME :- SANYUKTA NANDKUMAR GADGE

TASK -6 :- Sales Trend Analysis Using Aggregations

INTRODUCTION : Sales Trend Analysis using aggregations in the ONLINE_SALES database helps track and understand sales patterns over time. By summarizing data like total orders, revenue, and product performance, it reveals business trends, peak periods, and customer behavior to support data-driven decisions.

QUERY PLANNING

NUMBER OF OBJECTIVES:

1. Use Extract(Month From Order_Date) For Month
2. GROUP BY year/month.
3. Use SUM() for revenue.
4. COUNT(DISTINCT order_id) for volume.
5. Use ORDER BY for sorting.
6. Limit results for specific time periods.
7. Customer Purchase Summary (total spent by each customer)
8. Top 5 Best-Selling Products (by quantity)
9. Average Order Value
10. Category-wise Revenue Breakdown
11. Monthly Revenue Trend

1. QUERY CONSTRUCTION

Query 1: Use Extract(Month From Order_Date) For Month

```
SELECT EXTRACT(MONTH FROM order_date) AS order_month,
COUNT(*) AS total_orders FROM orders
GROUP BY EXTRACT(MONTH FROM order_date)
ORDER BY order_month;
```

Explaination : This Query Counts The Number Of Orders Placed In Each Month And Lists Them In Order From January To December.

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

order_month	total_orders
4	10

Query 2: GROUP BY year/month.

```
SELECT DATE_FORMAT(order_date, '%Y-%m-%d') AS order_date,
COUNT(*) AS total_orders, SUM(amount) AS total_sales
FROM orders
GROUP BY order_date
ORDER BY order_date;
```

Explaination: This query shows daily total orders and sales by grouping orders by date, counting how many orders happened each day, and summing up the total sales amount for each date.

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

order_date	total_orders	total_sales
2024-04-01	1	19999.99
2024-04-02	1	299.99
2024-04-03	1	1599.99
2024-04-07	1	49999.99
2024-04-08	1	1299.99
2024-04-10	1	1299.99
2024-04-20	1	999.99
2024-04-23	1	799.99
2024-04-28	1	1899.99
2024-04-29	1	899.99

Query 3: Use SUM() for revenue.

```
SELECT DATE_FORMAT(order_date, '%Y-%m-%d') AS order_date,

SUM(amount) AS total_revenue

FROM orders

GROUP BY order_date

ORDER BY order_date;
```

Explanation : This query calculates total revenue per day by summing the **amount** from the **orders** table, grouping by each order date, and listing the results in date order.

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

order_date	total_revenue
2024-04-01	19999.99
2024-04-02	299.99
2024-04-03	1599.99
2024-04-07	49999.99
2024-04-08	1299.99
2024-04-10	1299.99
2024-04-20	999.99
2024-04-23	799.99
2024-04-28	1899.99
2024-04-29	899.99

Query 4 : COUNT(DISTINCT order_id) for volume.

```
SELECT DATE_FORMAT(order_date, '%Y-%m') AS order_date,

COUNT(DISTINCT order_id) AS order_volume

FROM order GROUP BY order_date ORDER BY order_date;
```

Explaination : This query shows the number of unique orders placed each month by formatting the order date to year-month, counting distinct order IDs, grouping by month, and ordering the results chronologically.

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

order_date	total_revenue
2024-04-01	19999.99
2024-04-02	299.99
2024-04-03	1599.99
2024-04-07	49999.99
2024-04-08	1299.99
2024-04-10	1299.99
2024-04-20	999.99
2024-04-23	799.99
2024-04-28	1899.99
2024-04-29	899.99

Query 5: Use ORDER BY for sorting.

```
SELECT DATE_FORMAT(order_date, '%Y-%m') AS order_date,

SUM(amount) AS total_revenue,COUNT(DISTINCT order_id) AS order_volume

FROM orders

GROUP BY order_date

ORDER BY total_revenue DESC;
```

Explanation : This query shows monthly total revenue and number of orders, sorted by highest revenue first.

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

order_date	total_revenue	order_volume
2024-04	49999.99	1
2024-04	19999.99	1
2024-04	1899.99	1
2024-04	1599.99	1
2024-04	1299.99	1
2024-04	1299.99	1
2024-04	999.99	1
2024-04	899.99	1
2024-04	799.99	1
2024-04	299.99	1

Query 6: Limit results for specific time periods.

```
SELECT DATE_FORMAT(order_date, '%Y-%m') AS order_date,

SUM(amount) AS total_revenue,

COUNT(DISTINCT order_id) AS order_volume

FROM orders

WHERE order_date BETWEEN '2024-04-01' AND '2024-04-30'

GROUP BY order_date

ORDER BY total_revenue DESC;
```

Explanation: The query calculates the total revenue and distinct order count for each day in April 2024, groups by date, and sorts the results by revenue in descending order. It uses GROUP BY 1 to group by the first column (order_date) in the SELECT statement.

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

order_date	total_revenue	order_volume
2024-04	49999.99	1
2024-04	19999.99	1
2024-04	1899.99	1
2024-04	1599.99	1
2024-04	1299.99	1
2024-04	1299.99	1
2024-04	999.99	1
2024-04	899.99	1
2024-04	799.99	1
2024-04	299.99	1

Query 7: Customer Purchase Summary (total spent by each customer)

```
SELECT c.customer_name, SUM(od.price * od.quantity) AS total_spent
FROM customers c
JOIN orders o ON c.customer_id = o.order_id
JOIN order_details od ON o.order_id = od.order_id
GROUP BY c.customer_name
ORDER BY total_spent DESC;
```

Explanation: The query calculates each customer's total spending by joining the customers, orders, and order_details tables, then sorts customers by total spending in descending order.

Result Grid | Filter Rows:

customer_name	total_spent
Rohit Sawant	49999.99
Saloni Patil	20799.84
sanyukta gadge	19999.98
Swapnil Mane	19999.75
Saujanya Gadge	11699.91
Sunny Jadhav	7599.96
Sahil Shelar	6399.96
Kausthubh Tandel	3999.96
pragati gadge	3599.96
sakshi badhe	299.99

Query 8: Top 5 Best-Selling Products (by quantity)

```
SELECT p.product_name, SUM(od.quantity) AS total_sold

FROM order_details od

JOIN products p ON od.product_id = p.product_id

GROUP BY p.product_name

ORDER BY total_sold DESC

LIMIT 5;
```

Explanation : The query finds the top 5 best-selling products by quantity. It joins the order_details and products tables, sums the quantities sold for each product, groups by product name, orders by the total quantity sold in descending order, and limits the result to the top 5 products.

Result Grid | Filter Rows:

product_name	total_sold
Shoes	5
SQL	4
Headphones	3
Book: Java Basics	2
Kitchen Mixer Grinder	2

Query 9 : Average Order Value

```
SELECT AVG(order_total) AS avg_order_value

FROM ( SELECT o.order_id, SUM(od.quantity * od.price) AS order_total

    FROM orders o

    JOIN order_details od ON o.order_id = od.order_id

    GROUP BY o.order_id

) AS order_totals;
```

Explanation : This query calculates the **average order value** by first calculating the total value of each order (sum of quantity * price), then finding the average of those totals

Result Grid | Filter Rows:

avg_order_value
14439.930000

Query 10: Category-wise Revenue Breakdown

```
SELECT p.category, SUM(od.quantity * od.price) AS category_revenue

FROM order_details od

JOIN products p ON od.product_id = p.product_id

GROUP BY p.category

ORDER BY category_revenue DESC;
```

Explanation : This query calculates the total revenue for each product category by multiplying quantity and price from the order_details table, then joining it with the products table. It groups the results by category and sorts them by revenue in descending order.

Result Grid | Filter Rows:

category	category_revenue
Electronics	95699.80
Book	24699.79
Footwear	19999.75
Clothes	3999.96

Query 11: Monthly Revenue Trend

```
SELECT DATE_FORMAT(o.order_date, '%Y-%m') AS month,

    SUM(od.quantity * od.price) AS monthly_revenue

FROM orders o

JOIN order_details od ON o.order_id = od.order_id

GROUP BY month

ORDER BY month;
```

Ecplanation: The query calculates the monthly revenue by multiplying the quantity and price from the order_details table. It joins the orders and order_details tables, groups by month, and orders the results by month.

Result Grid | Filter Rows:

month	monthly_revenue
2024-04	144399.30

ERROR HANDLING & PERFORMANCE CONSIDERATION:

Error Handling

1. **File Access Issues**: Always check if the file exists and is accessible before reading; handle exceptions like FileNotFoundError, PermissionError, or IOError.
2. **Corrupted or Unsupported Format**: Use try-catch blocks when parsing PDFs to handle corrupted files or unsupported formats gracefully.
3. **Text Extraction Failures**: Not all PDFs have extractable text (e.g., scanned images). Handle these cases with fallback logic or OCR as needed.

Performance Considerations

1. **Lazy Loading and Page Range Processing**: Process PDFs page-by-page or in chunks instead of loading the entire file into memory.
2. **Use Efficient Libraries**: Choose libraries like PyMuPDF (fitz) or pdfplumber known for faster and efficient text extraction over older ones like PyPDF2.
3. **Avoid Repeated Parsing**: If accessing the same PDF multiple times, cache the parsed results to avoid redundant computation.