SQL_DATA_ANALYTICS_PROJECT

```
/*
=============================================================
Create Database and Schemas
=============================================================
USE master;
GO

-- Drop and recreate the 'DataWarehouseAnalytics' database
IF EXISTS (SELECT 1 FROM sys.databases WHERE name = 'DataWarehouseAnalytics')
BEGIN
   ALTER DATABASE DataWarehouseAnalytics SET SINGLE_USER WITH ROLLBACK
IMMEDIATE;
   DROP DATABASE DataWarehouseAnalytics;
END;
GO

-- Create the 'DataWarehouseAnalytics' database
CREATE DATABASE DataWarehouseAnalytics;
GO

USE DataWarehouseAnalytics;
GO

-- Create Schemas

CREATE SCHEMA gold;
GO
```

```
CREATE TABLE gold.dim_customers(
        customer_key int,
        customer_id int,
        customer_number nvarchar(50),
        first_name nvarchar(50),
        last_name nvarchar(50),
        country nvarchar(50),
        marital_status nvarchar(50),
        gender nvarchar(50),
        birthdate date,
        create_date date
);
GO

CREATE TABLE gold.dim_products(
        product_key int ,
        product_id int ,
        product_number nvarchar(50) ,
        product_name nvarchar(50) ,
        category_id nvarchar(50) ,
        category nvarchar(50) ,
        subcategory nvarchar(50) ,
        maintenance nvarchar(50) ,
        cost int,
        product_line nvarchar(50),
        start_date date
);
GO
```

```
CREATE TABLE gold.fact_sales(
	order_number nvarchar(50),
	product_key int,
	customer_key int,
	order_date date,
	shipping_date date,
	due_date date,
	sales_amount int,
	quantity tinyint,
	price int
);
GO

TRUNCATE TABLE gold.dim_customers;
GO

BULK INSERT gold.dim_customers
FROM 'C:\Users\Amit Mishra\Desktop\SQLClass\sql-data-analytics-project\datasets\csv-files\gold.dim_customers.csv'
WITH (
	FIRSTROW = 2,
	FIELDTERMINATOR = ',',
	TABLOCK
);
GO

TRUNCATE TABLE gold.dim_products;
GO
```

```
BULK INSERT gold.dim_products
FROM 'C:\Users\Amit Mishra\Desktop\SQLClass\sql-data-analytics-project\datasets\csv-files\gold.dim_products.csv'
WITH (
	FIRSTROW = 2,
	FIELDTERMINATOR = ',',
	TABLOCK
);
GO

TRUNCATE TABLE gold.fact_sales;
GO

BULK INSERT gold.fact_sales
FROM 'C:\Users\Amit Mishra\Desktop\SQLClass\sql-data-analytics-project\datasets\csv-files\gold.fact_sales.csv'
WITH (
	FIRSTROW = 2,
	FIELDTERMINATOR = ',',
	TABLOCK
);
GO

-- Analyse sales performance over time
-- Quick Date Functions
SELECT
   YEAR(order_date) AS order_year,
   MONTH(order_date) AS order_month,
   SUM(sales_amount) AS total_sales,
   COUNT(DISTINCT customer_key) AS total_customers,
```

```
    SUM(quantity) AS total_quantity
FROM gold.fact_sales
WHERE order_date IS NOT NULL
GROUP BY YEAR(order_date), MONTH(order_date)
ORDER BY YEAR(order_date), MONTH(order_date);

-- DATETRUNC()
SELECT
    DATETRUNC(month, order_date) AS order_date,
    SUM(sales_amount) AS total_sales,
    COUNT(DISTINCT customer_key) AS total_customers,
    SUM(quantity) AS total_quantity
FROM gold.fact_sales
WHERE order_date IS NOT NULL
GROUP BY DATETRUNC(month, order_date)
ORDER BY DATETRUNC(month, order_date);

-- FORMAT()
SELECT
    FORMAT(order_date, 'yyyy-MMM') AS order_date,
    SUM(sales_amount) AS total_sales,
    COUNT(DISTINCT customer_key) AS total_customers,
    SUM(quantity) AS total_quantity
FROM gold.fact_sales
WHERE order_date IS NOT NULL
GROUP BY FORMAT(order_date, 'yyyy-MMM')
ORDER BY FORMAT(order_date, 'yyyy-MMM');
```

Results | Messages

	order_year	order_month	total_sales	total_customers	total_quantity
1	2010	12	43419	14	14
2	2011	1	469795	144	144
3	2011	2	466307	144	144
4	2011	3	485165	150	150
5	2011	4	502042	157	157
6	2011	5	561647	174	174
7	2011	6	737793	230	230
8	2011	7	596710	188	188
9	2011	8	614516	193	193
10	2011	9	603047	185	185
11	2011	10	708164	221	221
12	2011	11	660507	208	208
13	2011	12	669395	222	222
14	2012	1	495363	252	252
15	2012	2	506992	260	260
16	2012	3	373478	212	212
17	2012	4	400324	219	219
18	2012	5	358866	207	207
19	2012	6	555142	318	318
20	2012	7	444533	246	246
21	2012	8	523887	294	294
22	2012	9	486149	269	269
23	2012	10	535125	313	313
24	2012	11	537918	324	324
25	2012	12	624454	354	483
26	2013	1	857758	627	1677
27	2013	2	771218	1373	3454
28	2013	3	1049732	1631	4087
29	2013	4	1045860	1564	3979
30	2013	5	1284456	1719	4400
31	2013	6	1642948	1948	5025
32	2013	7	1371595	1796	4673
33	2013	8	1545910	1898	4848
34	2013	9	1447324	1832	4616
35	2013	10	1673261	2073	5304
36	2013	11	1780688	2036	5224
37	2013	12	1874128	2133	5520
38	2014	1	45642	834	1970

Query executed successfully.

```
/*
=============================================================================
====
Cumulative Analysis
=============================================================================
====
Purpose:
   - To calculate running totals or moving averages for key metrics.
   - To track performance over time cumulatively.
   - Useful for growth analysis or identifying long-term trends.

SQL Functions Used:
   - Window Functions: SUM() OVER(), AVG() OVER()
=============================================================================
====
*/

-- Calculate the total sales per month
-- and the running total of sales over time
SELECT
        order_date,
        total_sales,
        SUM(total_sales) OVER (ORDER BY order_date) AS running_total_sales,
        AVG(avg_price) OVER (ORDER BY order_date) AS moving_average_price
FROM
(
   SELECT
      DATETRUNC(year, order_date) AS order_date,
```

```
        SUM(sales_amount) AS total_sales,
        AVG(price) AS avg_price
    FROM gold.fact_sales
    WHERE order_date IS NOT NULL
    GROUP BY DATETRUNC(year, order_date)
) t



/*
=============================================================================
Performance Analysis (Year-over-Year, Month-over-Month)
=============================================================================
Purpose:
    - To measure the performance of products, customers, or regions over time.
    - For benchmarking and identifying high-performing entities.
    - To track yearly trends and growth.

SQL Functions Used:
    - LAG(): Accesses data from previous rows.
    - AVG() OVER(): Computes average values within partitions.
    - CASE: Defines conditional logic for trend analysis.
=============================================================================
*/

/* Analyze the yearly performance of products by comparing their sales
to both the average sales performance of the product and the previous year's sales */
```

```
WITH yearly_product_sales AS (
    SELECT
        YEAR(f.order_date) AS order_year,
        p.product_name,
        SUM(f.sales_amount) AS current_sales
    FROM gold.fact_sales f
    LEFT JOIN gold.dim_products p
        ON f.product_key = p.product_key
    WHERE f.order_date IS NOT NULL
    GROUP BY
        YEAR(f.order_date),
        p.product_name
)
SELECT
    order_year,
    product_name,
    current_sales,
    AVG(current_sales) OVER (PARTITION BY product_name) AS avg_sales,
    current_sales - AVG(current_sales) OVER (PARTITION BY product_name) AS diff_avg,
    CASE
        WHEN current_sales - AVG(current_sales) OVER (PARTITION BY product_name) > 0
THEN 'Above Avg'
        WHEN current_sales - AVG(current_sales) OVER (PARTITION BY product_name) < 0
THEN 'Below Avg'
        ELSE 'Avg'
    END AS avg_change,
    -- Year-over-Year Analysis
    LAG(current_sales) OVER (PARTITION BY product_name ORDER BY order_year) AS
py_sales,
    current_sales - LAG(current_sales) OVER (PARTITION BY product_name ORDER BY
order_year) AS diff_py,
```

```
    CASE
        WHEN current_sales - LAG(current_sales) OVER (PARTITION BY product_name ORDER BY order_year) > 0 THEN 'Increase'
        WHEN current_sales - LAG(current_sales) OVER (PARTITION BY product_name ORDER BY order_year) < 0 THEN 'Decrease'
        ELSE 'No Change'
    END AS py_change
FROM yearly_product_sales
ORDER BY product_name, order_year;


-- Which categories contribute the most to overall sales.


WITH category_sales AS (
SELECT category,SUM(sales_amount) AS total_sales
FROM gold.fact_sales f
JOIN gold.dim_products p
ON f.product_key = p.product_key
GROUP BY category
)
SELECT category,total_sales,
SUM(total_sales) OVER() AS overall_sales,
CONCAT(ROUND((CAST (total_sales AS FLOAT) / SUM(total_sales)  OVER())*100,2),'%') AS Total_percentage
FROM category_sales
ORDER BY total_sales DESC


/* Segment products into cost ranges and
how many products fall into each segment */
```

```
WITH product_segment AS(
SELECT product_key,product_name,cost,
CASE WHEN cost < 100 THEN 'Below 100'
    WHEN cost BETWEEN 100 AND 500 THEN '100-500'
        WHEN cost BETWEEN 500 AND 1000 THEN '500-1000'
        ELSE 'Above 1000'
END cost_range
FROM gold.dim_products
)

SELECT cost_range ,
COUNT(product_key) AS Total_Product
FROM product_segment
GROUP BY cost_range
ORDER BY Total_Product DESC


/*Group customers into three segments based on their spending behavior:
        - VIP: Customers with at least 12 months of history and spending more than €5,000.
        - Regular: Customers with at least 12 months of history but spending €5,000 or less.
        - New: Customers with a lifespan less than 12 months.
And find the total number of customers by each group
*/

WITH customer_spending AS(
SELECT c.customer_key,SUM(f.sales_amount) AS total_spending,
MIN(order_date) AS first_order,
MAX(order_date) AS last_order,
DATEDIFF(month,MIN(order_date),MAX(order_date)) AS lifespan
```

```
FROM gold.dim_customers c
JOIN gold.fact_sales f
ON f.customer_key = c.customer_key
GROUP BY c.customer_key
)
SELECT customer_segment,COUNT(customer_key) AS total_customers
FROM(
SELECT customer_key,total_spending,lifespan,
CASE WHEN lifespan >= 12 AND total_spending > 5000 THEN 'VIP'
     WHEN lifespan >= 12 AND total_spending <= 5000 THEN 'Regular'
          ELSE 'New'
END customer_segment
FROM customer_spending)t
GROUP BY customer_segment
ORDER BY total_customers DESC;

/*
===============================================================================
Customer Report
===============================================================================
Purpose:
   - This report consolidates key customer metrics and behaviors

Highlights:
   1. Gathers essential fields such as names, ages, and transaction details.
        2. Segments customers into categories (VIP, Regular, New) and age groups.
   3. Aggregates customer-level metrics:
          - total orders
```

- total sales

- total quantity purchased

- total products

- lifespan (in months)

4. Calculates valuable KPIs:

- recency (months since last order)

- average order value

- average monthly spend

```
================================================================================
====
*/


--
================================================================================
==
-- Create Report: gold.report_customers
--
================================================================================
==
CREATE VIEW gold.report_customers AS

WITH base_query AS(
/*-----------------------------------------------------------------------------
1) Base Query: Retrieves core columns from tables
-----------------------------------------------------------------------------*/
SELECT
f.order_number,
f.product_key,
f.order_date,
f.sales_amount,
f.quantity,
```

```
c.customer_key,
c.customer_number,
CONCAT(c.first_name, ' ', c.last_name) AS customer_name,
DATEDIFF(year, c.birthdate, GETDATE()) age
FROM gold.fact_sales f
LEFT JOIN gold.dim_customers c
ON c.customer_key = f.customer_key
WHERE order_date IS NOT NULL)

, customer_aggregation AS (
/*---------------------------------------------------------------------------
2) Customer Aggregations: Summarizes key metrics at the customer level
---------------------------------------------------------------------------*/
SELECT
    customer_key,
    customer_number,
    customer_name,
    age,
    COUNT(DISTINCT order_number) AS total_orders,
    SUM(sales_amount) AS total_sales,
    SUM(quantity) AS total_quantity,
    COUNT(DISTINCT product_key) AS total_products,
    MAX(order_date) AS last_order_date,
    DATEDIFF(month, MIN(order_date), MAX(order_date)) AS lifespan
FROM base_query
GROUP BY
    customer_key,
    customer_number,
    customer_name,
```

```
        age
)
SELECT
customer_key,
customer_number,
customer_name,
age,
CASE
        WHEN age < 20 THEN 'Under 20'
        WHEN age between 20 and 29 THEN '20-29'
        WHEN age between 30 and 39 THEN '30-39'
        WHEN age between 40 and 49 THEN '40-49'
        ELSE '50 and above'
END AS age_group,
CASE
   WHEN lifespan >= 12 AND total_sales > 5000 THEN 'VIP'
   WHEN lifespan >= 12 AND total_sales <= 5000 THEN 'Regular'
   ELSE 'New'
END AS customer_segment,
last_order_date,
DATEDIFF(month, last_order_date, GETDATE()) AS recency,
total_orders,
total_sales,
total_quantity,
total_products
lifespan,
-- Compuate average order value (AVO)
CASE WHEN total_sales = 0 THEN 0
        ELSE total_sales / total_orders
```

```
END AS avg_order_value,
-- Compuate average monthly spend
CASE WHEN lifespan = 0 THEN total_sales
    ELSE total_sales / lifespan
END AS avg_monthly_spend
FROM customer_aggregation;

SELECT * FROM gold.report_customers;

/*
===============================================================================
Product Report
===============================================================================
Purpose:
   - This report consolidates key product metrics and behaviors.

Highlights:
   1. Gathers essential fields such as product name, category, subcategory, and cost.
   2. Segments products by revenue to identify High-Performers, Mid-Range, or Low-Performers.
   3. Aggregates product-level metrics:
      - total orders
      - total sales
      - total quantity sold
      - total customers (unique)
      - lifespan (in months)
   4. Calculates valuable KPIs:
      - recency (months since last sale)
```

```
    - average order revenue (AOR)
    - average monthly revenue
================================================================================
*/
--
================================================================================
-- Create Report: gold.report_products
--
================================================================================
IF OBJECT_ID('gold.report_products', 'V') IS NOT NULL
  DROP VIEW gold.report_products;
GO

CREATE VIEW gold.report_products AS

WITH base_query AS (
/*-------------------------------------------------------------------------
1) Base Query: Retrieves core columns from fact_sales and dim_products
-------------------------------------------------------------------------*/
  SELECT
        f.order_number,
    f.order_date,
            f.customer_key,
    f.sales_amount,
    f.quantity,
    p.product_key,
    p.product_name,
    p.category,
```

```
        p.subcategory,
        p.cost
    FROM gold.fact_sales f
    LEFT JOIN gold.dim_products p
        ON f.product_key = p.product_key
    WHERE order_date IS NOT NULL  -- only consider valid sales dates
),

product_aggregations AS (
/*---------------------------------------------------------------------------
2) Product Aggregations: Summarizes key metrics at the product level
---------------------------------------------------------------------------*/
SELECT
    product_key,
    product_name,
    category,
    subcategory,
    cost,
    DATEDIFF(MONTH, MIN(order_date), MAX(order_date)) AS lifespan,
    MAX(order_date) AS last_sale_date,
    COUNT(DISTINCT order_number) AS total_orders,
        COUNT(DISTINCT customer_key) AS total_customers,
    SUM(sales_amount) AS total_sales,
    SUM(quantity) AS total_quantity,
        ROUND(AVG(CAST(sales_amount AS FLOAT) / NULLIF(quantity, 0)),1) AS
avg_selling_price
FROM base_query

GROUP BY
    product_key,
```

```
    product_name,
    category,
    subcategory,
    cost
)

/*---------------------------------------------------------------------------
  3) Final Query: Combines all product results into one output
---------------------------------------------------------------------------*/
SELECT
        product_key,
        product_name,
        category,
        subcategory,
        cost,
        last_sale_date,
        DATEDIFF(MONTH, last_sale_date, GETDATE()) AS recency_in_months,
        CASE
                WHEN total_sales > 50000 THEN 'High-Performer'
                WHEN total_sales >= 10000 THEN 'Mid-Range'
                ELSE 'Low-Performer'
        END AS product_segment,
        lifespan,
        total_orders,
        total_sales,
        total_quantity,
        total_customers,
        avg_selling_price,
        -- Average Order Revenue (AOR)
```

```
        CASE
            WHEN total_orders = 0 THEN 0
            ELSE total_sales / total_orders
        END AS avg_order_revenue,

        -- Average Monthly Revenue
        CASE
            WHEN lifespan = 0 THEN total_sales
            ELSE total_sales / lifespan
        END AS avg_monthly_revenue

FROM product_aggregations

SELECT * FROM gold.report_products;
```