Code Basics Resume Portfolio Challenge 9 – Retail Industry – SQL Challenge

Analyse Promotions and Provide Tangible Insights to Sales Director

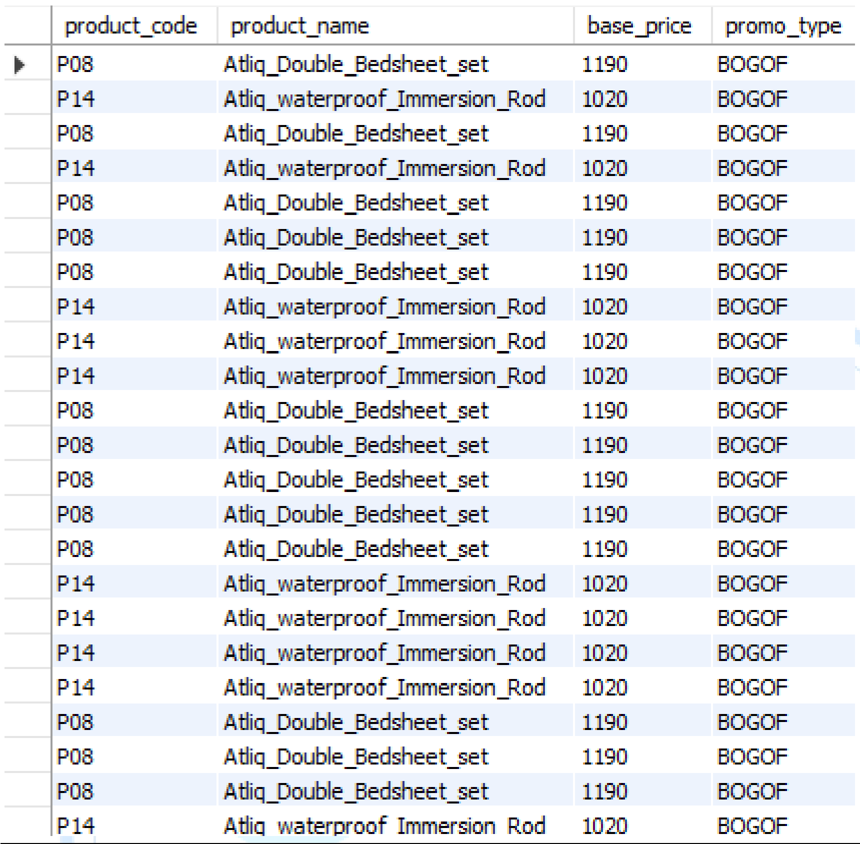
```
ALTER TABLE fact_events

CHANGE `quantity_sold(before_promo)` quantity_sold_before_promo INT;

ALTER TABLE fact_events

CHANGE `quantity_sold(after_promo)` quantity_sold_after_promo INT;

SELECT * FROM retail_events_db.fact_events;
```

----

1. Provide a list of products with a base price greater than 500 and that are featured in promo type of 'BOGOF' (Buy One Get One Free).This information will help us identify high-value products that are currently being heavily discounted, which can be useful for evaluating our pricing and promotion strategies.

```
SELECT
   p.product_code,
          p.product_name,
   f.base_price,
   f.promo_type
FROM
   dim_products as p
Join fact_events as f on f.product_code = p.product_code
WHERE
   base_price > 500
   AND promo_type = 'BOGOF';
```

Result :



product_code	product_name	base_price	promo_type
P08	Atliq_Double_Bedsheet_set	1190	BOGOF
P14	Atliq_waterproof_Immersion_Rod	1020	BOGOF
P08	Atliq_Double_Bedsheet_set	1190	BOGOF
P14	Atliq_waterproof_Immersion_Rod	1020	BOGOF
P08	Atliq_Double_Bedsheet_set	1190	BOGOF
P08	Atliq_Double_Bedsheet_set	1190	BOGOF
P08	Atliq_Double_Bedsheet_set	1190	BOGOF
P14	Atliq_waterproof_Immersion_Rod	1020	BOGOF
P14	Atliq_waterproof_Immersion_Rod	1020	BOGOF
P14	Atliq_waterproof_Immersion_Rod	1020	BOGOF
P08	Atliq_Double_Bedsheet_set	1190	BOGOF
P08	Atliq_Double_Bedsheet_set	1190	BOGOF
P08	Atliq_Double_Bedsheet_set	1190	BOGOF
P08	Atliq_Double_Bedsheet_set	1190	BOGOF
P08	Atliq_Double_Bedsheet_set	1190	BOGOF
P14	Atliq_waterproof_Immersion_Rod	1020	BOGOF
P14	Atliq_waterproof_Immersion_Rod	1020	BOGOF
P14	Atliq_waterproof_Immersion_Rod	1020	BOGOF
P14	Atliq_waterproof_Immersion_Rod	1020	BOGOF
P08	Atliq_Double_Bedsheet_set	1190	BOGOF
P08	Atliq_Double_Bedsheet_set	1190	BOGOF
P08	Atliq_Double_Bedsheet_set	1190	BOGOF
P14	Atliq waterproof Immersion Rod	1020	BOGOF

Explanation:

1. ***SELECT Clause:***
    - ***It specifies the columns that you want to retrieve in the result set.***
    - ***The selected columns are product_code and product_name from the dim_products table (aliased as p), and base_price and promo_type from the fact_events table (aliased as f).***
2. ***FROM Clause:***
    - ***It defines the tables involved in the query and their aliases.***
    - ***The query is using the dim_products table with the alias p and the fact_events table with the alias f.***
3. ***JOIN Clause:***
    - ***It specifies the condition for joining the two tables.***
    - ***The tables are joined on the condition where the product_code column in the fact_events table (f.product_code) matches the product_code column in the dim_products table (p.product_code).***

4. ***WHERE Clause:***

   - ***It filters the rows based on specified conditions.***
   - ***Rows are included in the result set only if the base_price is greater than 500 and the promo_type is 'BOGOF'.***

In summary, the query retrieves information about products from the dim_products table and corresponding events from the fact_events table where the base price is greater than 500 and the promotion type is 'BOGOF'. The JOIN clause ensures that the information is retrieved for matching product codes in both tables.

---

2. Generate a report that provides an overview of the number of stores in each city. The results will be sorted in descending order of store counts, allowing us to identify the cities with the highest store presence. The report includes two essential fields: city and store count, which will assist in optimizing our retail operations.

```
SELECT city, COUNT(store_id) AS store_count

FROM dim_stores

GROUP BY city

ORDER BY store_count DESC;
```

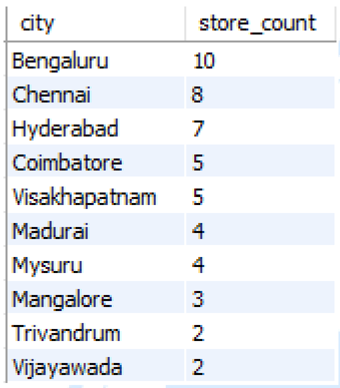
Explanation:

1. ***SELECT Clause:***
   - ***It specifies the columns that you want to retrieve in the result set.***
   - ***The query selects the city column directly and uses the COUNT(store_id) function to calculate the number of stores in each city. The result of this count is given an alias store_count.***
2. ***FROM Clause:***
   - ***It specifies the table from which the data will be retrieved.***
   - ***The query uses the dim_stores table as the source of the data.***
3. ***GROUP BY Clause:***
   - ***It groups the result set by one or more columns.***
   - ***In this case, the result set is grouped by the city column. This means that the count of stores will be calculated separately for each unique city.***
4. ***ORDER BY Clause:***

- ***It specifies the order in which the result set should be presented.***
- ***The query orders the result set by store_count in descending order (DESC), meaning the cities with the highest store count will appear first.***

In summary, this query retrieves a list of cities from the dim_stores table along with the count of stores in each city. The result set is grouped by city, and the order is determined by the store count in descending order.

Result :



city	store_count
Bengaluru	10
Chennai	8
Hyderabad	7
Coimbatore	5
Visakhapatnam	5
Madurai	4
Mysuru	4
Mangalore	3
Trivandrum	2
Vijayawada	2

---

3. Generate a report that displays each campaign along with the total revenue generated before and after the campaign? The report includes three key fields: campaign_name, total_revenue(before_promotion), total_revenue(after_promotion). This report should help in evaluating the financial impact of our promotional campaigns. (Display the values in millions)

```
WITH CampaignRevenueCTE AS (
  SELECT
    c.campaign_name,
    fe.base_price * fe.quantity_sold_before_promo AS revenue_before_promo,
    fe.base_price * fe.quantity_sold_after_promo AS revenue_after_promo
  FROM dim_campaigns c
  JOIN fact_events fe
  ON c.campaign_id = fe.campaign_id
)

SELECT
```

```
    campaign_name,
    SUM(revenue_before_promo) / 1000000 AS total_revenue_before_promotion,
    SUM(revenue_after_promo) / 1000000 AS total_revenue_after_promotion
FROM CampaignRevenueCTE
GROUP BY campaign_name;
```

Explanation :

***Common Table Expression (CTE) - CampaignRevenueCTE**:*

1. ***WITH** Clause:*
    - *It defines a Common Table Expression (CTE) named **CampaignRevenueCTE**.*
    - *The CTE consists of a SELECT statement that retrieves data from two tables: **dim_campaigns** (aliased as **c**) and **fact_events** (aliased as **fe**).*
    - *The JOIN condition links the **campaign_id** from **dim_campaigns** to the **campaign_id** from **fact_events**.*
    - *The SELECT statement calculates two derived columns:*
        - ***revenue_before_promo**: The product of **base_price** and **quantity_sold_before_promo**.*
        - ***revenue_after_promo**: The product of **base_price** and **quantity_sold_after_promo**.*

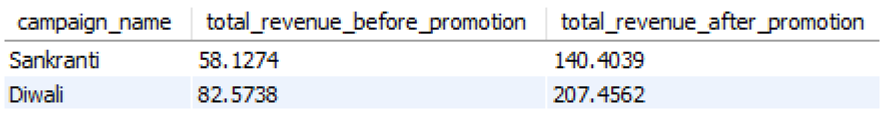
***Main Query**: 2. **SELECT** Clause:*

- *It selects columns for the final result set.*
- *The selected columns include **campaign_name** from the CTE.*

3. ***SUM** and Division:*
    - *The query uses the **SUM** function to calculate the total revenue before and after promotion for each campaign.*
    - *The result is divided by 1,000,000 to represent the total revenue in millions.*
4. ***FROM** Clause:*
    - *It specifies the source of the data, which is the previously defined CTE (**CampaignRevenueCTE**).*
5. ***GROUP BY** Clause:*
    - *It groups the result set by the **campaign_name** column.*

*In summary, this SQL code calculates the revenue metrics (before and after promotion) for each campaign by creating a Common Table Expression (**CampaignRevenueCTE**). The main query then* *aggregates this information, presenting the total revenue before and after promotion in millions, grouped by the campaign name.*

Result:



campaign_name	total_revenue_before_promotion	total_revenue_after_promotion
Sankranti	58.1274	140.4039
Diwali	82.5738	207.4562

4.Produce a report that calculates the Incremental Sold Quantity (ISU%) for each category during the Diwali campaign. Additionally, provide rankings for the categories based on their ISU%. The report will include three key fields: category, isu%, and rank order. This information will assist in assessing the category-wise success and impact of the Diwali campaign on incremental sales.

```
WITH DiwaliCategoryMetrics AS (
  SELECT
    dp.category,
    SUM(fe.quantity_sold_after_promo - fe.quantity_sold_before_promo) AS
incremental_sold_quantity,
    SUM(fe.quantity_sold_before_promo) AS baseline_sold_quantity
  FROM dim_campaigns dc
  JOIN fact_events fe ON dc.campaign_id = fe.campaign_id
  JOIN dim_products dp ON fe.product_code = dp.product_code
  WHERE dc.campaign_name = 'Diwali'
  GROUP BY dp.category
)

SELECT
  category,
  COALESCE(
    (CAST(incremental_sold_quantity AS DECIMAL(10, 2)) / NULLIF(baseline_sold_quantity, 0)) *
100,
    0
  ) AS isu_percentage,
    RANK() OVER (ORDER BY (CAST(incremental_sold_quantity AS DECIMAL(10, 2)) /
NULLIF(baseline_sold_quantity, 0)) DESC) AS rank_order

FROM DiwaliCategoryMetrics

ORDER BY rank_order;
```

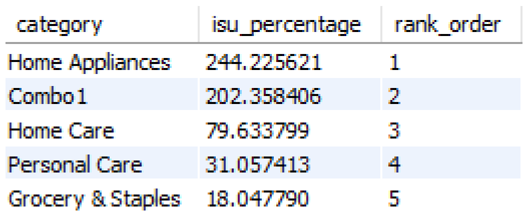
Explanation :

1. ***Common Table Expression (CTE) - DiwaliCategoryMetrics:***
    - ***This CTE calculates metrics related to sold quantities for each product category during the Diwali campaign.***
    - ***It selects the category column along with the calculated columns:***
        - ***incremental_sold_quantity: Sum of the difference between quantity_sold_after_promo and quantity_sold_before_promo.***
        - ***baseline_sold_quantity: Sum of quantity_sold_before_promo.***
    - ***The data is retrieved by joining the dim_campaigns table (dc alias), fact_events table (fe alias), and dim_products table (dp alias) based on the appropriate keys (campaign_id and product_code).***
2. ***SELECT Clause (Main Query):***
    - ***It selects columns for the final result set, which includes the category, isu_percentage, and rank_order.***
3. ***COALESCE Function:***
    - ***It is used to handle potential division by zero errors.***
    - ***The isu_percentage is calculated as the percentage of incremental sold quantity relative to baseline sold quantity.***
    - ***The CAST function is used to ensure that the result is in the format DECIMAL(10, 2).***
    - ***The result is multiplied by 100 to express the percentage.***
4. ***RANK() OVER Clause:***
    - ***It assigns a rank to each category based on the calculated ISU percentage in descending order.***
    - ***The RANK() function is applied to the result set ordered by ISU percentage in descending order.***
5. ***ORDER BY Clause (Main Query):***
    - ***It orders the final result set by the rank_order column.***

In summary, this SQL code calculates and presents the incremental sold unit (ISU) percentage for each product category during the Diwali campaign. The result is ordered by the rank assigned to each category based on the ISU percentage in descending order. The COALESCE function is used to

handle potential division by zero errors. The CAST function is used for data type formatting, and the RANK() OVER clause is used for ranking.

Result :



category	isu_percentage	rank_order
Home Appliances	244.225621	1
Combo1	202.358406	2
Home Care	79.633799	3
Personal Care	31.057413	4
Grocery & Staples	18.047790	5

---

5. Create a report featuring the Top 5 products, ranked by Incremental Revenue Percentage (IR%), across all campaigns. The report will provide essential information including product name, category, and ir%. This analysis helps identify the most successful products in terms of incremental revenue across our campaigns, assisting in product optimization.

```
SELECT
    dp.product_name,
    dp.category,
    COALESCE(
        (SUM(fe.base_price * fe.quantity_sold_after_promo) - SUM(fe.base_price *
fe.quantity_sold_before_promo)) /
        NULLIF(SUM(fe.base_price * fe.quantity_sold_before_promo), 0) * 100,
        0
    ) AS ir_percentage
FROM dim_products dp
JOIN fact_events fe ON dp.product_code = fe.product_code
GROUP BY dp.product_name, dp.category
ORDER BY ir_percentage DESC
LIMIT 5;
```

Explanation :

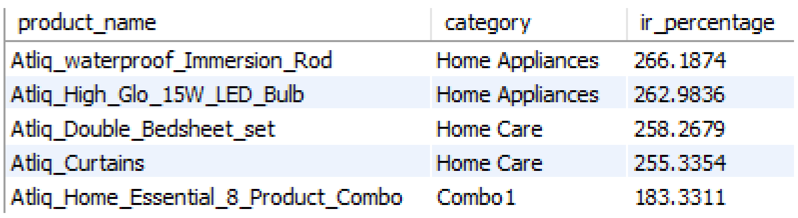
1. SELECT Clause:
    - It specifies the columns to be included in the result set.

- The selected columns are product_name and category from the dim_products table (aliased as dp).
- It calculates the improvement rate percentage using the COALESCE function to handle potential division by zero errors.

2. COALESCE Function:
    - It is used to handle division by zero scenarios. If the denominator (SUM of base price * quantity sold before promo) is zero, it returns 0 to avoid errors.
    - The result is multiplied by 100 to get a percentage.
3. FROM Clause:
    - It specifies the tables and their aliases involved in the query.
    - The query joins the dim_products table (dp) with the fact_events table (fe) on the product_code column.
4. GROUP BY Clause:
    - It groups the result set by product_name and category.
    - This is important for aggregating the quantities and base prices for each product in the calculation.
5. ORDER BY Clause:
    - It orders the result set by the calculated improvement rate percentage (ir_percentage) in descending order (DESC).
6. LIMIT Clause:
    - It restricts the result set to only the top 5 rows, giving you the products with the highest improvement rate percentages.

In summary, this SQL code calculates and retrieves the top 5 products with the highest improvement rate percentages based on the quantity sold before and after promotions. The improvement rate is calculated and expressed as a percentage. The COALESCE function is used to handle potential division by zero errors.

Result :



product_name	category	ir_percentage
Atliq_waterproof_Immersion_Rod	Home Appliances	266.1874
Atliq_High_Glo_15W_LED_Bulb	Home Appliances	262.9836
Atliq_Double_Bedsheet_set	Home Care	258.2679
Atliq_Curtains	Home Care	255.3354
Atliq_Home_Essential_8_Product_Combo	Combo1	183.3311

Presented by :

Souvanik Chaudhury