

Retail Intelligence with SQL

*Advanced Analytics for **Kuwait's Hypermarkets***

(A weekend project by Bilal Jamaludeen)

Project Brief:

This project is designed to showcase **advanced SQL skills** in a real-world retail environment, particularly in Kuwait's hypermarket sector. By simulating typical business scenarios faced by large-scale retail and e-commerce platforms (like **Grand Hyper**), this project aims to provide valuable insights using **complex SQL techniques** such as **CTEs, window functions**, and **advanced aggregation**.

Key Focus Areas:

- **Sales Optimization:** Analyzed product sales data to identify top-selling products across different branches and regions.
- **Stock Management:** Detecting low-stock, high-demand products to ensure timely restocking and efficient inventory management.
- **Customer Behavior:** Understanding customer shopping patterns, identifying one-time buyers, and flagging loyalty candidates for targeted marketing.
- **Promotions:** Analyzing the effectiveness of promo codes, identifying discount-driven customers, and leveraging this information for personalized marketing strategies.
- **Revenue Insights:** Determining peak revenue days and correlating customer activity with specific days of the week to inform promotional strategies.
- **Forecasting Needs:** Predicting restocking needs for top-selling products using rolling averages, ensuring the availability of high-demand goods.

SQL Techniques Used:

- **CTEs (Common Table Expressions)** to organize and simplify complex queries.
- **Window Functions** (e.g., `ROW_NUMBER()`, `RANK()`, `AVG()`) for analyzing trends across multiple rows.
- **Aggregation & Grouping** to gain deep insights from transactional data.
- **Date Functions** to perform time-based analyses and identify trends over time.

Business Impact:

By applying advanced SQL queries, this project helps retail managers and analysts in **Kuwait's hypermarkets** unlock actionable insights for:

- **Strategic decision-making**
- **Improving operational efficiency**
- **Enhancing customer experience**

This project highlights practical skills in **data analytics** and demonstrates how SQL can be leveraged to make data-driven decisions in the fast-paced retail industry.

Database Structure Overview

The project is designed around a retail/e-commerce database schema. It includes the essential tables that provide insights into customer behavior, sales performance, inventory management, and promotional activities.

1. Customers Table

This table contains details about each customer, helping track their purchasing behavior.

Column Name	Data Type	Description
customer_id	INT	Unique identifier for each customer.
first_name	VARCHAR(100)	Customer's first name.
last_name	VARCHAR(100)	Customer's last name.
email	VARCHAR(255)	Customer's email address.
signup_date	DATE	Date when the customer signed up.
phone	VARCHAR(15)	Customer's phone number.

2. Products Table

This table tracks the products available for sale, their details, and stock information.

Column Name	Data Type	Description
product_id	INT	Unique identifier for each product.
product_name	VARCHAR(255)	Name of the product.

category	VARCHAR(100)	Category of the product (e.g., Electronics, Apparel).
price	DECIMAL(10,2)	Price of the product.
stock_qty	INT	Current stock level of the product.
created_at	DATE	Date when the product was added.

3. Sales Table

This table records individual sales transactions, capturing customer purchases.

Column Name	Data Type	Description
sale_id	INT	Unique identifier for each sale.
customer_id	INT	Foreign key linking to the Customers table.
product_id	INT	Foreign key linking to the Products table.
quantity	INT	Quantity of the product sold.
sale_date	DATE	Date when the sale occurred.
total_price	DECIMAL(10,2)	Total price of the transaction.
promo_code	VARCHAR(50)	Promo code used, if applicable.

4. Inventory Table

This table tracks inventory updates, including restocking events.

Column Name	Data Type	Description
inventory_id	INT	Unique identifier for inventory changes.
product_id	INT	Foreign key linking to the Products table.
quantity	INT	Quantity of the product restocked.
action	VARCHAR(50)	Type of inventory action (e.g., "Restock", "Return").
action_date	DATE	Date when the action took place.

5. Promo_Codes Table

This table stores details of promo codes and discount campaigns.

Column Name	Data Type	Description
promo_id	INT	Unique identifier for each promo code.
promo_code	VARCHAR(50)	Promo code used in sales.
discount	DECIMAL(5,2)	Discount percentage offered by the promo.
valid_from	DATE	Start date of the promo code's validity.
valid_to	DATE	End date of the promo code's validity.

6. Branches Table

This table contains information about different store branches across the retail chain.

Column Name	Data Type	Description
branch_id	INT	Unique identifier for each branch.
branch_name	VARCHAR(100)	Name of the branch.
city	VARCHAR(100)	City where the branch is located.
region	VARCHAR(100)	Regional location of the branch.

Relationships between Tables

- **Customers ↔ Sales:** One-to-many relationship. Each customer can have multiple sales records.
- **Sales ↔ Products:** Many-to-one relationship. Each sale is linked to a specific product.
- **Products ↔ Inventory:** One-to-many relationship. Products can have multiple inventory updates (restocking).
- **Promo_Codes ↔ Sales:** One-to-many relationship. A promo code can be applied to multiple sales transactions.
- **Branches ↔ Sales:** Many-to-one relationship. Each sale is associated with a specific branch (though this could be inferred by the branch's geographic data).

Data Flow Overview

1. **Customer Activity:** Customers make purchases recorded in the **Sales** table. Each sale includes details about the products bought, quantities, total price, and whether a promo code was applied.
2. **Inventory Management:** Products and inventory levels are updated in the **Inventory** table whenever stock is replenished. This helps track product availability and restocking needs.
3. **Promo Codes:** Promo codes are applied to sales transactions, allowing the system to track which customers are using discounts and which products are frequently bought with discounts.
4. **Sales Trends:** By analyzing sales data, the business can understand top-selling products, customer purchasing behavior, and trends across branches and regions.
5. **Branch Analysis:** The **Branches** table helps understand sales by location, enabling performance evaluations across different store locations.

This database structure is designed to support a range of analytical queries that can help improve decision-making for sales, marketing, inventory, and customer retention strategies.

Business-driven SQL query questions

1. Identify Top-Selling Products in Each Branch During the Last 30 Days
2. Find Items Consistently Low in Stock but High in Demand
3. Detect Promo-Code Users Who Only Shop with Discounts
4. Identify One-Time Buyers Across All Branches
5. Flag Customers Who Shop Every Week (Loyalty Candidates)

Top-Selling Products in Each Branch During the Last 30 Days

This query provides insights into the best-selling products across different branches, helping with inventory and sales strategies.

```
○ ○ ○ Retail_Store_Data

1 -- Top-Selling Products in Each Branch During the Last 30 Days --
2 WITH top_products AS (
3     SELECT
4         p.product_id,
5         p.product_name,
6         b.branch_name,
7         SUM(s.quantity) AS total_quantity_sold
8     FROM sales s
9     JOIN products p ON s.product_id = p.product_id
10    JOIN branches b ON s.branch_id = b.branch_id
11    WHERE s.sale_date ≥ CURDATE() - INTERVAL 30 DAY
12    GROUP BY p.product_id, b.branch_name
13 )
14 SELECT branch_name, product_name, total_quantity_sold
15 FROM top_products
16 ORDER BY branch_name, total_quantity_sold DESC;
```

Explanation:

- **CTE (Common Table Expression):** The query starts with a CTE called `top_products` that aggregates the total quantity of each product sold at each branch in the last 30 days.
- **SUM(s.quantity):** Sums the number of units sold for each product at each branch.
- **WHERE clause:** Filters the sales to only include transactions within the last 30 days.
- **ORDER BY:** Orders the result by `branch_name` and the total quantity sold in descending order, so the top-selling products come first for each branch.

Items Consistently Low in Stock but High in Demand

This query identifies products that are selling well but have low stock levels. It helps businesses identify high-demand items that are running out of stock so they can restock before losing sales.

```
○ ○ ○ Retail_Store_Data

1 -- Items Consistently Low in Stock but High in Demand --
2 WITH low_stock_demand AS (
3     SELECT
4         p.product_id,
5         p.product_name,
6         SUM(s.quantity) AS total_quantity_sold,
7         i.stock_qty
8     FROM sales s
9     JOIN products p ON s.product_id = p.product_id
10    JOIN inventory i ON s.product_id = i.product_id
11    WHERE s.sale_date ≥ CURDATE() - INTERVAL 30 DAY
12    GROUP BY p.product_id
13 )
14 SELECT product_name, total_quantity_sold, stock_qty
15 FROM low_stock_demand
16 WHERE stock_qty < 10 AND total_quantity_sold > 50
17 ORDER BY total_quantity_sold DESC;
18
```

Explanation:

- **CTE (Common Table Expression):** The query starts with a CTE called `low_stock_demand` that aggregates the total quantity sold for each product in the last 30 days, along with the current stock quantity for each product.
- **SUM(s.quantity):** Sums the number of units sold for each product over the last 30 days, capturing the total demand.
- **WHERE clause:** Filters the data to only include sales transactions from the last 30 days, and the condition ensures that products with low stock (less than 10 units) and high demand (more than 50 units sold) are selected.
- **ORDER BY:** The results are ordered by the total quantity sold in descending order, ensuring the products that are in highest demand but low in stock appear first.

Promo-Code Users Who Only Shop with Discounts

This query helps businesses identify customers who only make purchases when using promo codes, allowing them to target this group with personalized marketing strategies.

It also helps in assessing the effectiveness of promo codes and discounts in driving sales.

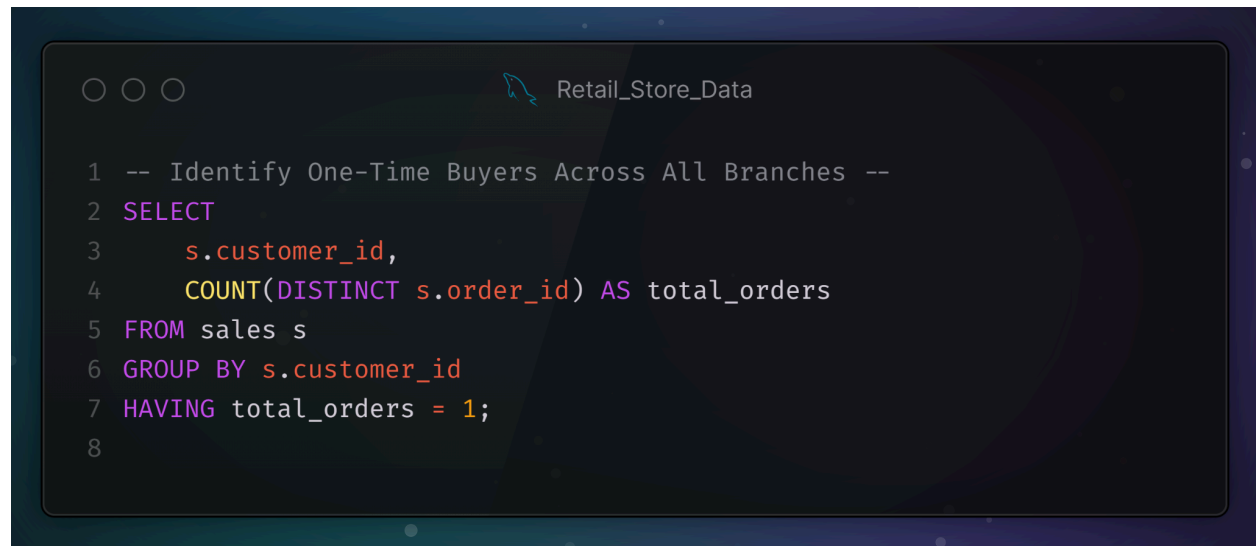
```
○ ○ ○ Retail_Store_Data

1 -- Detect Promo-Code Users Who Only Shop with Discounts --
2 WITH promo_code_users AS (
3     SELECT
4         s.customer_id,
5         COUNT(DISTINCT s.order_id) AS total_orders_with_promo,
6         SUM(CASE WHEN s.discount_code IS NOT NULL THEN 1 ELSE 0 END)
7     AS orders_with_discount,
8         COUNT(DISTINCT s.order_id) - SUM(CASE WHEN s.discount_code
9     IS NOT NULL THEN 1 ELSE 0 END) AS orders_without_discount
10    FROM sales s
11    GROUP BY s.customer_id
12 )
13 SELECT
14     p.customer_id,
15     p.total_orders_with_promo,
16     p.orders_with_discount,
17     p.orders_without_discount
18 FROM promo_code_users p
19 WHERE p.orders_with_discount = p.total_orders_with_promo
20     AND p.orders_without_discount = 0;
```

Explanation The query returns a list of customers who only make purchases when using discounts or promo codes. This helps businesses identify customers who rely on discounts to make purchases.

Identify One-Time Buyers Across All Branches

The query returns a list of customers who have made exactly one purchase across all branches, helping businesses identify customers who might be harder to retain and could benefit from targeted marketing or promotions to encourage repeat purchases.

A screenshot of a code editor window titled 'Retail_Store_Data'. The editor shows a SQL query with line numbers 1 through 8. The query is designed to identify one-time buyers by selecting customer IDs and counting their unique orders, then filtering for those with exactly one order.

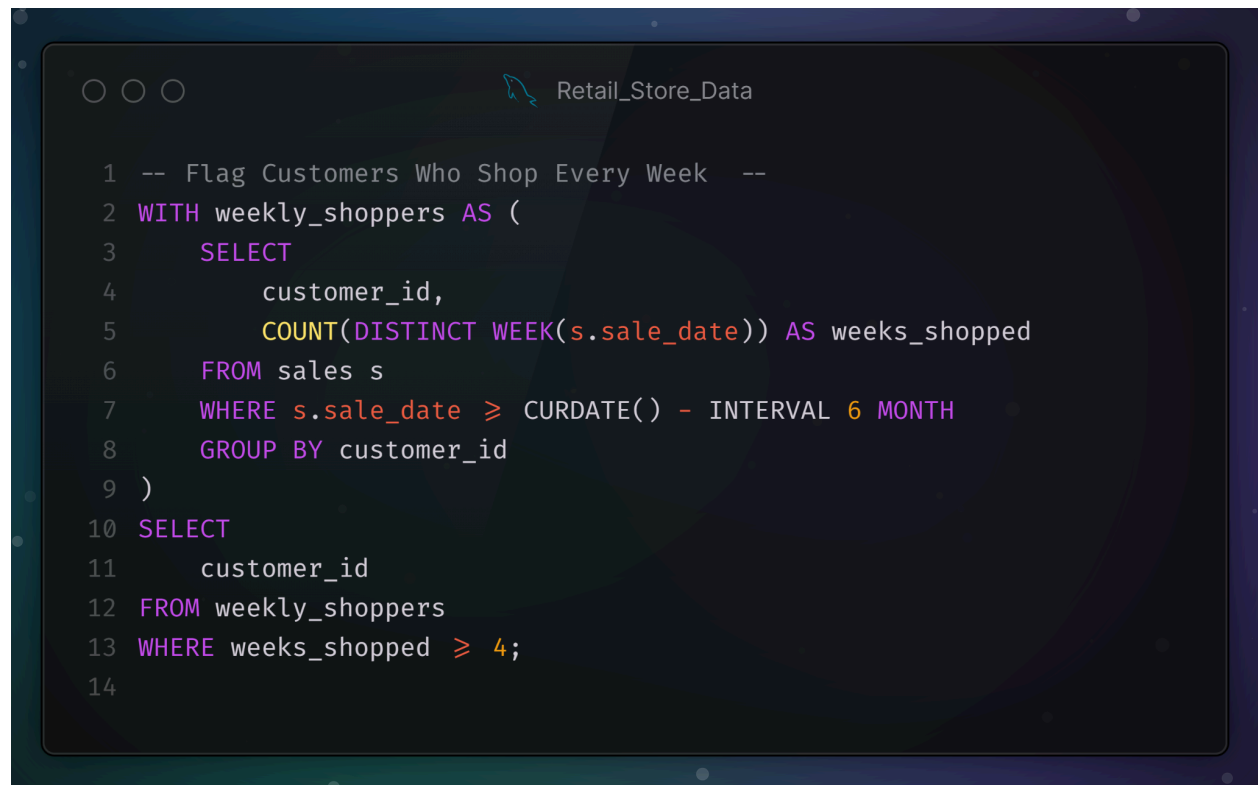
```
1 -- Identify One-Time Buyers Across All Branches --
2 SELECT
3     s.customer_id,
4     COUNT(DISTINCT s.order_id) AS total_orders
5 FROM sales s
6 GROUP BY s.customer_id
7 HAVING total_orders = 1;
8
```

Explanation

- **SELECT:** The query selects the `customer_id` and counts the total number of unique orders placed by each customer (`COUNT(DISTINCT s.order_id)`).
- **GROUP BY:** The data is grouped by `customer_id`, which means the total orders are calculated for each individual customer.
- **HAVING clause:** The `HAVING total_orders = 1` condition filters out customers who have placed more than one order. This ensures that only one-time buyers (customers who made exactly one purchase) are included in the result.

Flag Customers Who Shop Every Week (Loyalty Candidates)

The query identifies customers who shop consistently on a weekly basis. These customers are good candidates for loyalty programs or targeted marketing to keep them engaged and increase their lifetime value.



```
1 -- Flag Customers Who Shop Every Week --
2 WITH weekly_shoppers AS (
3     SELECT
4         customer_id,
5         COUNT(DISTINCT WEEK(s.sale_date)) AS weeks_shopped
6     FROM sales s
7     WHERE s.sale_date ≥ CURDATE() - INTERVAL 6 MONTH
8     GROUP BY customer_id
9 )
10 SELECT
11     customer_id
12 FROM weekly_shoppers
13 WHERE weeks_shopped ≥ 4;
14
```

Explanation

- CTE (Common Table Expression): The query begins with a CTE called `weekly_shoppers` that identifies customers who have made purchases within the last 6 months.
 - It counts how many distinct weeks each customer has made purchases in (using `COUNT(DISTINCT WEEK(s.sale_date))`), which helps identify if they are shopping consistently every week.
- WHERE clause: It filters for customers who have made purchases in at least 4 distinct weeks over the past 6 months (`weeks_shopped ≥ 4`), which flags them as potential loyal customers or loyalty candidates.

This SQL project focuses on solving key business problems within the retail and e-commerce industry by leveraging data analysis techniques such as CTEs, window functions, and aggregations. Each query is designed to provide valuable insights that can help businesses improve operations, optimize inventory, increase sales, and enhance customer retention strategies.

- By identifying top-selling products, businesses can make better stock management decisions.
- The low-stock, high-demand analysis helps prevent stockouts of popular items, ensuring that sales opportunities are not lost.
- Recognizing promo-code users who only shop with discounts helps businesses assess their pricing strategy and target customers who rely heavily on discounts.
- Finding one-time buyers across branches allows businesses to focus on retention strategies for customers who haven't made repeat purchases.
- Flagging loyalty candidates (customers who shop consistently every week) helps businesses design loyalty programs to retain high-value customers.

Each of these queries addresses practical challenges faced by retailers and e-commerce businesses and provides actionable insights to improve both customer satisfaction and business performance. By applying such analyses, businesses can make more informed, data-driven decisions that align with both short-term goals and long-term growth strategies.