**2023**

**CS-2005 Database Systems**

**National University of Computer and Emerging Sciences**

**Assignment # 02**

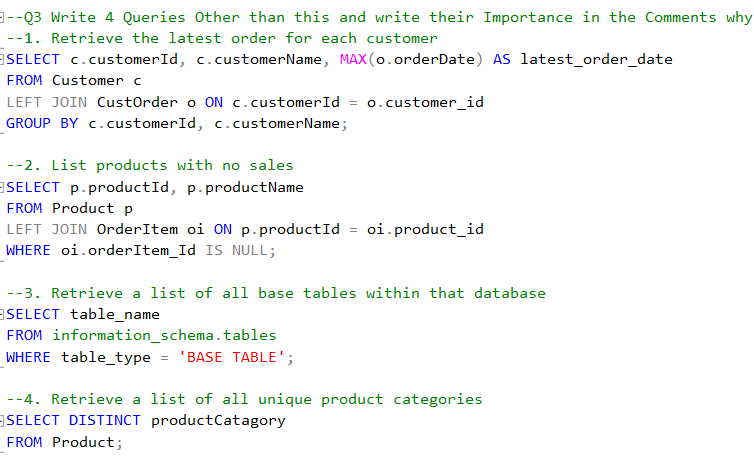
**Arsh Malik & Mahareeb Fatima (21i-2752 & 21i-1564)**

**SQL Queries**

**Implementation and Explanation**

1. **Write 4 Queries Other than this and write their Importance in the Comments why do you think they are important and where can they be used.**

**Query:**

****

**Output:**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Explanation:**

Query 1 retrieves the latest order for each customer. This query is useful when you want to find out the most recent order placed by each customer. It can be used to identify customers who haven’t placed an order in a while and may need to be contacted to encourage them to make a purchase.

Query 2 lists the products with no sales which is helpful when you want to identify products that have not been sold yet. It can be used to determine which products may need additional marketing or promotion to increase sales.

Query 3 retrieves a list of all base tables within that database which is useful when you want to get a list of all tables in a database that are not views or system tables. It can be used to identify the structure of a database and understand how different tables are related.

Query 4 retrieves a list of all unique product categories which is helpful when you want to get a list of all unique product categories in your database. It can be used to understand how many different types of products you have and how they are categorized.

1. **List the top 5 customers who spent the most money.**

**Query:**

**A white background with black and green text

Description automatically generated**

**Output:**

A screenshot of a computer

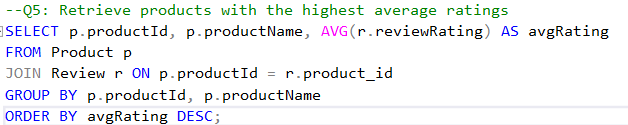
Description automatically generated

**Explanation:**

Since we were supposed to print the top 5 customers, we used the **TOP** function. The **MAX** function is used to calculate the maximum order total amount for each customer, which is then sorted in descending order 1.

1. **Retrieve products with the highest average ratings**

**Query:**

****

**Output:**

A screenshot of a computer

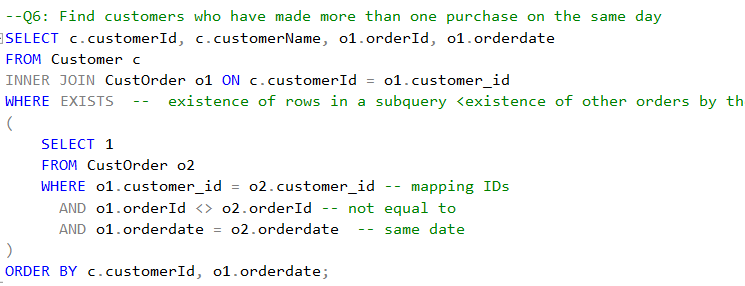
Description automatically generated

**Explanation:**

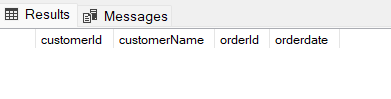
Since we were asked to retrieve the products with highest average rating, we used the **AVG** function to calculate the average review rating for each product, and then ordered the output in descending order .

1. **Find customers who have made more than one purchase on the same day.**

**Query:**

****

**Output:**



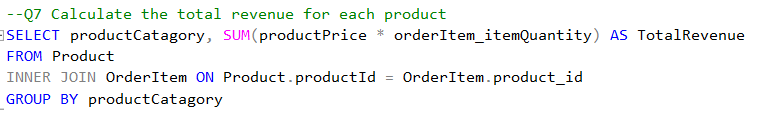
**Explanation:**

The requirement for this query was to retrieve customers who made more than 1 purchase on the same date. For this we joined the Customer and CustOrder tables and then using a subquery, we checked for the existence of other orders by the same customer on the same date. The **EXISTS** clause is used to filter out customers who have another order with a different order ID but the same order date .

***Note:*** *In our database, no customers ordered more than once on the same date.*

1. **Calculate the total revenue for each product category.**

**Query:**

****

**Output:**

A screenshot of a computer

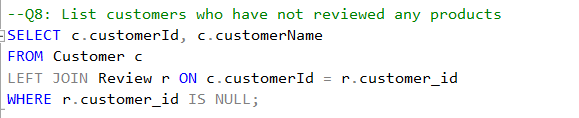
Description automatically generated

**Explanation:**

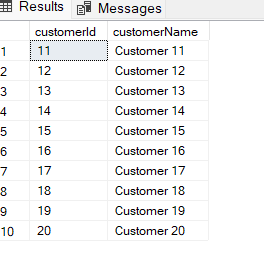
The requirement for this query was to calculate the total revenue for each product category. For this we joined the Product and OrderItem tables and used the **SUM** function to calculate the total revenue by multiplying the order item quantity and product price for each product. For better display, we used the **GROUP** clause to print the results by productCatagory.

1. **List customers who have not reviewed any products.**

**Query:**



**Output:**

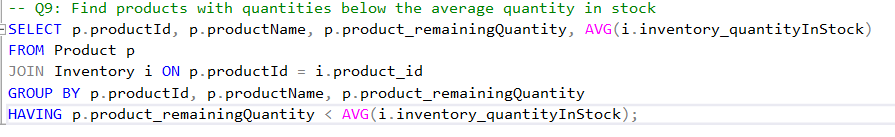


**Explanation:**

The requirement for this query was to list the customers who haven't yet reviewed any products. We did this by performing a left join between the Customer and Review tables. The **LEFT JOIN** clause is used to include all customers from the Customer table, even if they do not have a corresponding review in the Review table. The **WHERE** clause filters out customers who have a null value in the customer\_id column of the Review table.

1. **Find products with quantities below the average quantity in stock.**

**Query:**



**Output:**

A screenshot of a computer

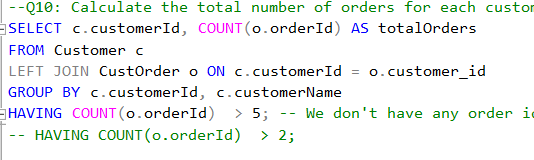
Description automatically generated

**Explanation:**

The requirement for this query was to identify products with stock quantities below the average. To do this we joined the Product and Inventory tables, calculated the average quantity in stock using the **AVG** function, and compared it with the remaining quantity of each product using the **HAVING** clause to filter for products with lower quantities

1. **Calculate the total number of orders for each customer and show only those with more than 5 orders.**

**Query:**



**Output:**

A screenshot of a computer

Description automatically generated

**Explanation:**

The requirement for this query was to calculate the total number of orders for each customer and then filter out those customers who had more than 5 orders. To do this we joined the Customer and CustOrder tables on customerId. We used the **COUNT** function to calculate the total number of orders for each customer, and then filtered them using the **HAVING** clause to show only those customers with more than 5 orders.

***Note:*** *Our database had no customers with more than five orders.*

1. **Retrieve the 3 most recent orders for a specific customer.**

**Query:**

A close-up of a computer screen

Description automatically generated

**Output:**

A screenshot of a computer

Description automatically generated

**Explanation:**

The requirement for this query was to retrieve the three most recent orders for a specific customer. To do this we used the **WHERE** clause to filter out the customer (in our case customerid = 1). We also used the **TOP** clause to select the 3 most recent orders for customer ID 1 and sorted it in descending order by orderdate.

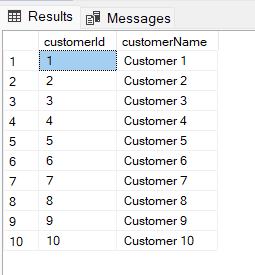
1. **List customers who have purchased products from at least two different sellers.**

**Query:**

A screen shot of a computer code

Description automatically generated

**Output:**



**Explanation:**

The requirement for this query was to retrieve customers who purchased products from at least two different sellers. To do this we joined the Product and CustOrder tables on productId and customerId. We used the **COUNT** function to calculate the number of **distinct** sellers for each customer, and then filtered only those customers who have purchased products from at least two different sellers using the **WHERE** clause to show only those customers

1. **Find customers who have placed an order in the last 30 days.**

**Query:**

A white background with black and pink text

Description automatically generated

**Output:**

A screenshot of a computer

Description automatically generated

**Explanation:**

The requirement for this query was to identify customers who had placed orders in the last 30 days. To do this we joined the Customer and CustOrder tables on customerId and used the **WHERE** clause to filter out orders that were older than 30 days from the current date.

1. **List customers who have made a purchase in every product category.**

**Query:**

A screenshot of a computer code

Description automatically generated

**Output:**

A screenshot of a computer

Description automatically generated

**Explanation:**

The requirement for this query was to identify customers who had placed orders in the last 30 days. To do this we joined the Customer and CustOrder tables on customerId and used the **WHERE** clause to filter out orders that were older than 30 days from the current date.

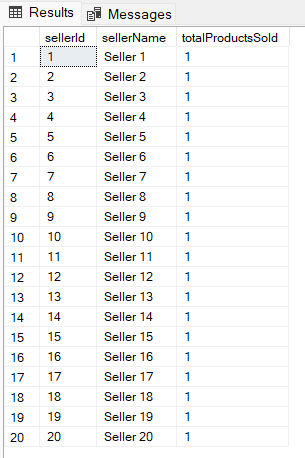
1. **Calculate the total number of products sold by each seller.**

**Query:**

A screen shot of a computer code

Description automatically generated

**Output:**

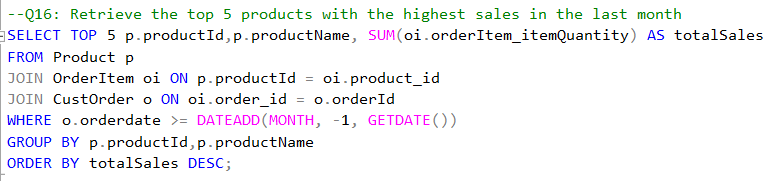


**Explanation:**

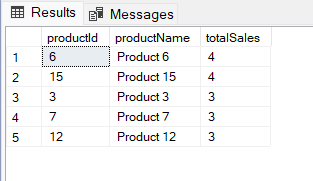
The requirement for this query was to calculate the total number of products sold by each seller. To do this we joined the Seller, Product, and OrderItem tables on sellerId, productId, and product\_id, and used the **COUNT** function to calculate the total number of distinct products sold by each seller. We also used the **ORDER BY** clause to present the results in descending order based on the total number of products sold.

1. **Retrieve the top 5 products with the highest sales in the last month.**

**Query:**



**Output:**

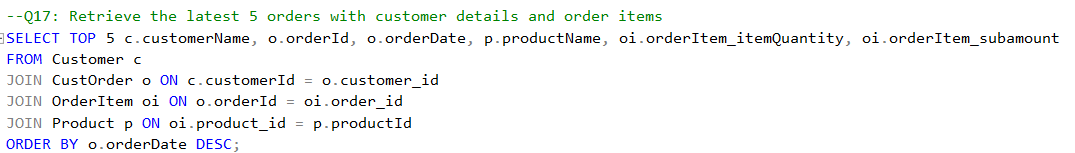


**Explanation:**

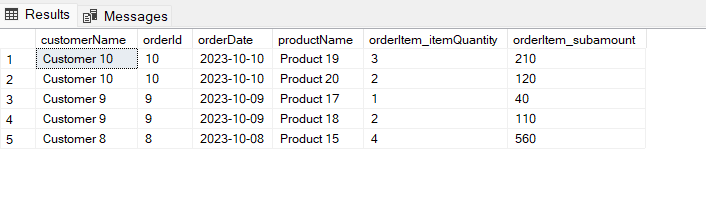
The requirement for this query was to retrieve the top 5 products with the highest sales in the last month. To do this we joined the Product, OrderItem, and CustOrder tables on productId, product\_id, and orderId, and used the **SUM** function to calculate the total sales for each product by multiplying the order item quantity and product price for each product. We also used the **ORDER BY** clause to present the results in descending order based on the total sales.

1. **Retrieve the latest 5 orders along with customer details and order items for each order.**

**Query:**



**Output:**



**Explanation:**

The requirement for this query was to retrieve the latest 5 orders along with customer details and order items for each order. To do this we joined the Customer, CustOrder, OrderItem, and Product tables on customerId, orderId, order\_id, and productId. For this query, we selected the customer’s name, order ID, order date, product name, order item quantity, and order item subamount for each order and then used the **ORDER BY** clause to present the results in descending order based on the order date.

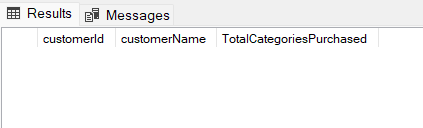
1. **Retrieve customers who have made purchases in every product category, along with the total number of categories they have purchased from.**

**Query:**

A screen shot of a computer code

Description automatically generated

**Output:**



**Explanation:**

The requirement for this query was to identify the customers who have made purchases in every product category. To do this we joined the Customer, CustOrder, OrderItem, and Product tables on customerId, orderId, order\_id, and productId and used the **COUNT** function along with the **DISTINCT** clause to calculate the number of **distinct** product categories purchased by each customer. After that, we used the **HAVING** clause to filter and show only those customers who have purchased products from all available product categories.

1. **List products that have never been reviewed and have quantities in stock greater than zero, along with the average rating for their category.**

**Query:**

A screenshot of a computer code

Description automatically generated

**Output:**

A screenshot of a computer

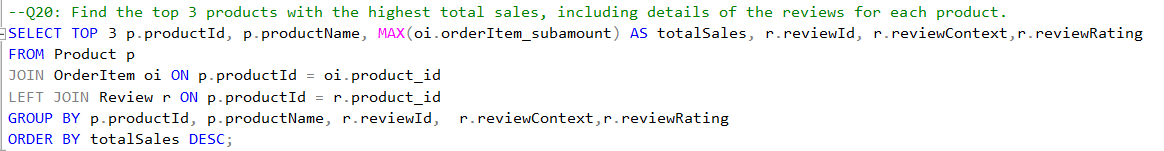
Description automatically generated

**Explanation:**

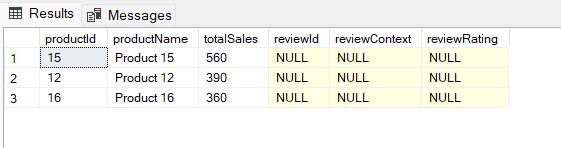
The requirement for this query was to retrieves products that have not been reviewed, have positive quantities in stock along with the average rating for their category. To do this we joined the Product and Review tables on productId and then used the WHERE clause to filter the products with positive remaining quantities and null review. We used the **AVG** function to calculate the average rating for each product category, and then grouped the display by product ID and name.

1. **Find the top 3 products with the highest total sales, including details of the reviews for each product.**

**Query:**



**Output:**

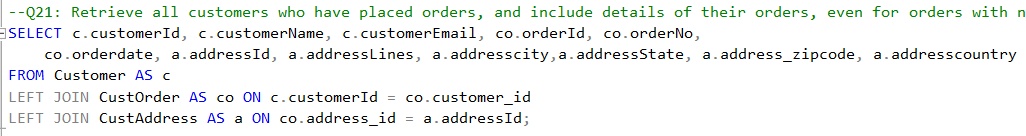


**Explanation:**

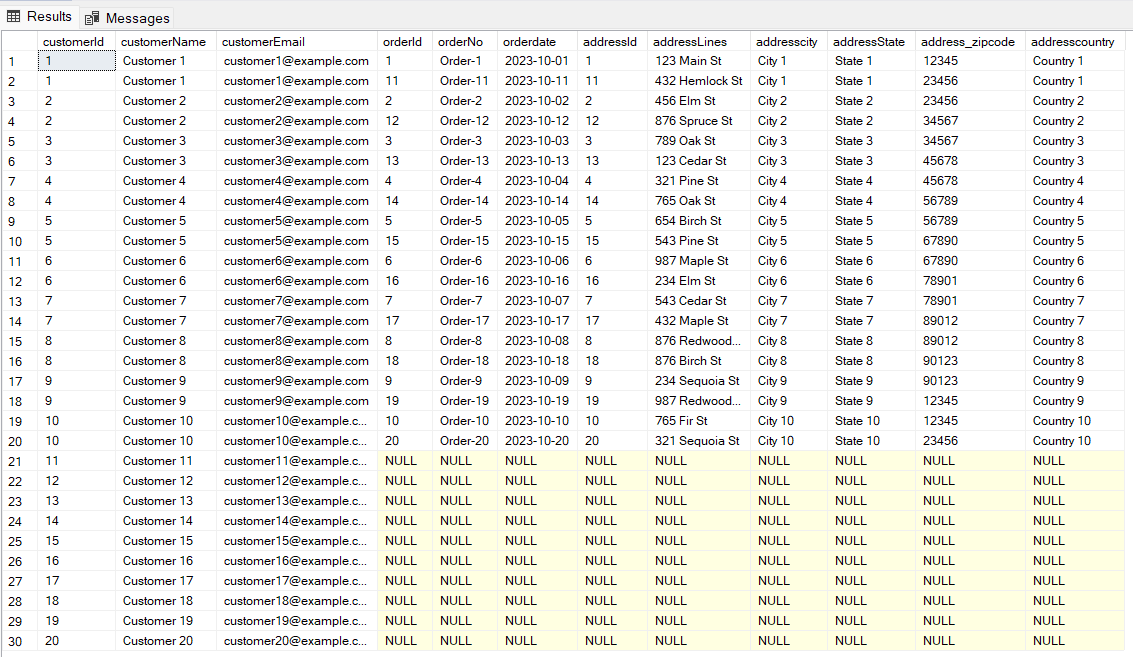
The requirement for this query was to identify the top 3 products with the highest total sales and include review details for each product. To do this we joined the Product, OrderItem, and Review tables on productId, product\_id, and product\_id, and then used the **GROUP BY** clause to group the results by product details and review information, along with the **ORDER BY** clause to order the products by their total sales in descending order to select the top 3 using the **TOP** clause. We used the **MAX** function to calculate the maximum order item subamount for each product.

1. **Retrieve all customers who have placed orders, and include details of their orders, even for orders with no associated customers. Include information about the shipping addresses for each order.**

**Query:**



**Output:**

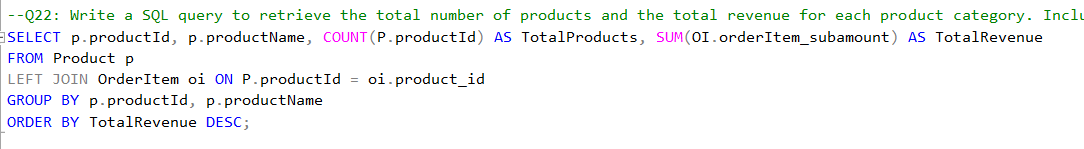


**Explanation:**

The requirement for this query was to retrieves information about all customers who have placed orders and includes details of their orders, even for orders with no associated customers (an order without an associated customer is something unheard of) along with the information about the shipping addresses for each order. To do this we used the **LEFT JOIN** clause joins between the Customer, CustOrder, and CustAddress tables, linking customers to their orders and orders to their associated addresses. The reason for using the **LEFT JOIN** clause was to include all customers from the Customer table, even if they do not have a corresponding order in the CustOrder table.

1. **Write a SQL query to retrieve the total number of products and the total revenue for each product category. Include products that may not have been sold. Additionally, order the product in descending order based on total revenue.**

**Query:**



**Output:**

A screenshot of a data

Description automatically generated

**Explanation:**

The requirement for this query was to calculate the total number of products and the total revenue for each product category, even including products that may not have been sold. To do this we used the **LEFT JOIN** clause joining the Product and OrderItem tables. The reason for using the **LEFT JOIN** clause was to include all products from the Product table, even if they do not have a corresponding order item in the OrderItem table. We used the **COUNT** function to calculate the total number of products for each product category, and the **SUM** function to calculate the total revenue for each product category. We also grouped the results by product details and ordered the products in descending order based on their total revenue.

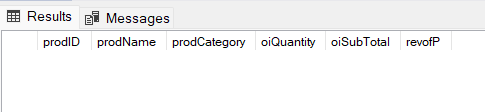
1. Write a SQL query to retrieve detailed information about products and their associated orders. Include the product ID, product name, product category, order quantity, order price, and total revenue for each product. Ensure that products that have not been sold are also included in the result. Filter the results to include only products in the 'Electronics' category with order quantities between 5 and 10. Additionally, order results in descending order based on total revenue.

**Query:**

A white background with black text

Description automatically generated

**Output:**



**Explanation:**

The requirement for this query was to retrieve product categories and their total number of products sold, displaying categories with more than 10 products sold. To do this we used the **LEFT JOIN** clause joining the Product and OrderItem table and then used the **GROUP BY** clause to group the results by productCatagory. Apart from that, we used the **HAVING** clause to filter the results to only include categories with more than 10 products sold. We did this by comparing the count of order items using the **COUNT** function. We used the **ORDER BY** clause at the end to sort the results in descending order by the total.

1. **Retrieve product categories with the total number of products sold, ordered in descending order by the total number of products sold, and show only categories with more than 10 products sold.**

**Query:**

A screenshot of a computer code

Description automatically generated

**Output:**

A screenshot of a computer message

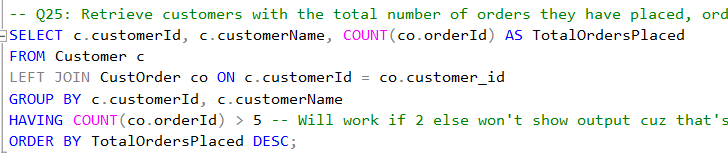
Description automatically generated

**Explanation:**

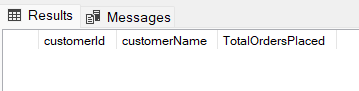
The requirement for this query was to retrieve product categories and their total number of products sold, displaying categories with more than 10 products sold. To do this we used the **LEFT JOIN** clause joining the Product and OrderItem table and then used the **GROUP BY** clause to group the results by productCatagory. Apart from that, we used the **HAVING** clause to filter the results to only include categories with more than 10 products sold. We did this by comparing the count of order items using the **COUNT** function. We used the **ORDER BY** clause at the end to sort the results in descending order by the total.

1. **Retrieve customers with the total number of orders they have placed, ordered in descending order by the total number of orders, and show only customers who have placed more than 5 orders.**

**Query:**



**Output:**



**Explanation:**

The requirement for this query was to retrieve customers along with the total number of orders they have placed, displaying the results in descending order based on the total number of orders with the restriction of showing only those customers who placed more than 5 orders. To do this we used the **LEFT JOIN** clause joining the Customer and CustOrder tables. We used the **COUNT** function is used to calculate the total number of orders placed by each customer, and then used the **GROUP BY** clause to group the result by customerid and custname along with the **HAVING** clause to filter out customers who have placed less than or equal to 5 orders. At the end, according to the instructions, we sorted the results in descending order by the total number of orders placed.