15 july 2021 -- Data Analysis with SQL

- 2 - TrUSE BikeStores

-- CROSS JOIN  
-- Write a query that returns the table to be used to add products that are in the Products table but not in the stocks table to the stocks table with quantity = 0.  
-- (Do not forget to add products to all stores.)  
-- Expected columns: store\_id, product\_id, quantity

SELECT B.store\_id, A.product\_id, A.product\_name, 0 quantity  
FROM production.products AS A  
CROSS JOIN sales.stores AS B  
WHERE A.product\_id NOT IN (SELECT product\_id FROM production.stocks)  
ORDER BY A.product\_id, B.store\_id

-- CROSS JOIN

-- Hangi markada hangi kategoride kaçar ürün olduğu bilgisine ihtiyaç duyuluyor  
-- Ürün sayısı hesaplamadan sadece marka \* kategori ihtimallerinin hepsini içeren bir tablo oluşturun.  
-- Çıkan sonucu daha kolay yorumlamak için brand\_id ve category\_id alanlarına göre sıralayın

SELECT \*  
FROM production.brands  
CROSS JOIN production.categories  
ORDER BY production.brands.brand\_id

-- SELF JOIN  
-- Write a query that returns the staff with their managers.  
-- Expected columns: staff first name, staff last name, manager name

SELECT \*  
FROM sales.staffs AS A  
JOIN sales.staffs AS B  
ON A.manager\_id = B.staff\_id

-- GROUPING OPERATIONS -1  
-- Write a query that checks if any product id is repeated in more than one row in the products table.

SELECT TOP 20 \*  
FROM production.products

SELECT A.product\_name, COUNT(A.product\_name)  
FROM production.products AS A  
GROUP BY A.product\_name  
HAVING COUNT(A.product\_name) >1;

-- WHERE is useful for another new table, for current table HAVING is okay.

SELECT product\_id, COUNT(product\_id) AS CNT\_PRODUCT  
FROM production.products  
GROUP BY product\_id, product\_name  
HAVING COUNT (product\_id) > 1;

SELECT product\_id, COUNT (\*) AS CNT\_PRODUCT  
FROM production.products  
GROUP BY product\_id  
HAVING COUNT (\*) > 1

-- GROUPING OPERATIONS -2  
-- Write a query that returns category ids with a maximum list price above 4000 or a minimum list price below 500.

SELECT category\_id, MAX(list\_price) AS max\_list\_price , MIN(list\_price) AS min\_list\_price  
FROM production.products  
GROUP BY category\_id  
HAVING MAX(list\_price)>4000 OR MIN(list\_price)<500;

-- GROUPING OPERATIONS -3  
-- Find the average product prices of the brands.  
-- As a result of the query, the average prices should be displayed in descending order.

SELECT A.brand\_name, AVG(B.list\_price) AS avg\_list\_price  
FROM production.brands AS A  
INNER JOIN production.products AS B  
ON A.brand\_id = B.brand\_id  
GROUP BY A.brand\_name  
ORDER BY AVG(B.list\_price) DESC;

SELECT A.brand\_name, AVG(B.list\_price) AS avg\_list\_price  
FROM production.brands AS A, production.products AS B  
WHERE A.brand\_id = B.brand\_id  
GROUP BY A.brand\_name  
ORDER BY avg\_list\_price DESC;  
-- As you can see, if you will write two table side by side with comma after

FROM expression, you can use WHERE instead of INNER JOIN

-- GROUPING OPERATIONS -4  
-- Write a query that returns BRANDS with an average product price of more than 1000.

SELECT B.brand\_name, AVG(list\_price) as avg\_price  
FROM production.products as A  
INNER JOIN production.brands as B  
ON A.brand\_id = B.brand\_id  
GROUP BY brand\_name  
HAVING AVG (list\_price) > 1000  
ORDER BY avg\_price ASC;

SELECT brands.brand\_name, AVG(products.list\_price) AS avg\_price  
FROM production.products, production.brands  
WHERE products.brand\_id = brands.brand\_id  
GROUP BY brands.brand\_name  
HAVING AVG(products.list\_price) > 1000  
ORDER BY AVG(products.list\_price) ASC;

-- GROUPING OPERATIONS -5  
-- Write a query that returns the net price paid by the customer for each order. (Don't neglect discounts and quantities)

SELECT A.order\_id, SUM(quantity \* list\_price \* (1-discount)) AS net\_value

-- (1-discount) for percentile  
FROM sales.order\_items AS A  
GROUP BY A.order\_id

SELECT order\_id, SUM(quantity \* (list\_price-list\_price\*discount)) AS net\_value

-- (1-discount) for percentile  
FROM sales.order\_items  
GROUP BY order\_id

-- Creatıng Summary Table Into Our Bıkestores Tables

SELECT \*  
INTO NEW\_TABLE  
FROM SOURCE\_TABLE  
WHERE ...

SELECT C.brand\_name as Brand, D.category\_name as Category, B.model\_year as Model\_Year,  
ROUND (SUM (A.quantity \* A.list\_price \* (1 - A.discount)), 0) total\_sales\_price  
INTO sales.sales\_summaryFROM sales.order\_items A, production.products B, production.brands C, production.categories D  
WHERE A.product\_id = B.product\_id  
AND B.brand\_id = C.brand\_id  
AND B.category\_id = D.category\_id  
GROUP BY  
C.brand\_name, D.category\_name, B.model\_year

-- GROUP BY with GROUPING SETS

-- 1. Total Sales (grouping by Brand)

SELECT SUM(total\_sales\_price)  
FROM sales.sales\_summary  
GROUP BY Brand-- 2. Total Sales (grouping by Category)

SELECT SUM(total\_sales\_price)  
FROM sales.sales\_summary  
GROUP BY Category-- 3. Total Sales (grouping by Brand and Category)

SELECT Brand, Category, SUM(total\_sales\_price)  
FROM sales.sales\_summary  
GROUP BY Brand, Category

-- 4. Total Sales (grouping by Brand and Category and Brand-Category with GROUPING SETS)

SELECT Brand, Category, SUM(total\_sales\_price)  
FROM sales.sales\_summary  
GROUP BY  
GROUPING SETS ((Brand),(Category),(Brand,Category),())

-- blank paranthesis is bringing us double null

ORDER BY 1,2;-- GROUP BY with ROLLUP-- 1. Total Sales (grouping by Brand and Category and Brand-Category with ROLLUP)

SELECT Brand, Category, SUM(total\_sales\_price)  
FROM sales.sales\_summary  
GROUP BY  
ROLLUP (Brand, Category)  
ORDER BY 1,2;-- GROUP BY with CUBE

-- 1. Total Sales (grouping by Brand and Category and Brand-Category with CUBE)

SELECT Brand, Category, SUM(total\_sales\_price)  
FROM sales.sales\_summary  
GROUP BY  
CUBE (Brand, Category)  
ORDER BY 1,2;