```
USE 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
-- CREATING SUMMARY TABLE INTO OUR BIKESTORES 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_summary
FROM 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;

CUBE her turlu kombinasyonu dokuyor, ROLLUP istenen duzeyde ihtiyac duyulan kombinasyonu dokuyor kisaca..

15.07.2021 DawSQL Sessinon 2
----- CROSS JOIN-----
-- Soru1: 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 brand_id
----- SELF JOIN------
-- Soru2: 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
SELECT A.first_name AS Staff_Name, A.last_name AS Staff_Last, B.first_name AS Manager
FROM sales.staffs A, sales.staffs B
WHERE A.manager_id = B.staff_id
---- GROUPBY / HAVING ----
--GROUPING OPERATION SORU1--
--Write a query that checks if any product id is repeated in more than one row in the products table.
SELECT A.product_name, COUNT(A.product_name)
FROM production.products AS A

```
GROUP BY A.product_name
HAVING COUNT(A.product_name) >1; --HAVING'DE kullandığın sütun Aggregate
te kullandığın sütun ismiyle aynı olmalı.
-- hocanın çözümü:
-- önce products ları görelim.
SELECT TOP 20*
FROM production.products
SELECT product_id, COUNT(*) AS CNT_PRODUCT
FROM production.products
GROUP BY
                product_id  -- bütün product_id lerin product tablosunda birer kere
geçtiğini gördüm.
SELECT product_id, COUNT(*) AS CNT_PRODUCT
FROM production.products
GROUP BY
                product_id
HAVING
                COUNT(*) > 1  --HAVING'DE kullandığın sütun Aggregate te
kullandığın sütun ismiyle aynı olmalı.
-- product_name e göre yapalım
SELECT product_name, COUNT(*) AS CNT_PRODUCT  -- count(*) tüm rowları say
demek. count(product_id) de aynı işi görür.
FROM production.products
GROUP BY
                product_name
HAVING
                COUNT (*) > 1
-- aşağıdaki gibi de kullanabiliriz.
SELECT product_name, COUNT(product_id) AS CNT_PRODUCT  -- count(*) tüm
rowları say demek. count(product_id) de aynı işi görür.
FROM production.products
GROUP BY
                product_name
HAVING
                COUNT (product_id) > 1
SELECT production_id, production_name, COUNT (*) CNT_PRODUCT
FROM production.products
GROUP BY
                product_name
HAVING
                COUNT (*) > 1
-- select te yazdığın sütunlar group by da olması gerekiyor. production_id group
by da olmadığı için hata verdi.
SELECT production_id, production_name, COUNT (*) CNT_PRODUCT
FROM production.products
GROUP BY
```

```
                product_name, product_id
HAVING
                COUNT (*) > 1
SELECT          product_id, COUNT (*) AS CNT_PRODUCT
FROM production.products
GROUP BY
                product_id
HAVING
                COUNT (*) > 1
--GROUPING OPERATION SORU 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, MIN(list_price) AS min_price, MAX(list_price) AS max_price
-- grupladığımız şey category_id olduğu için SELECT'te onu getiriyoruz
FROM production.products
-- ana tablo içinde herhangi bir kısıtlamam var mı yani where işlemi var mı? yok.
devam ediyorum
GROUP BY
                category_id
HAVING
                MIN(list_price) < 500 OR MAX(list_price) > 4000
--GROUPING OPERATION SORU 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_PRICE
FROM production.brands A, production.products B
-- buradaki virgül INNER JOIN ile aynı işi yapıyor! virgülle beraber WHERE
kullanıyoruz.
WHERE A.brand_id = B.brand_id
GROUP BY
                A.brand_name
ORDER BY
                AVG_PRICE DESC
-- (virgül + WHERE yerine--> INNER JOIN ile çözüm)
SELECT A.brand_name, AVG(B.list_price) AS AVG_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_PRICE DESC
-- ORDER BY 2 DESC olarak da yazabilirdik. Burada 2 --> SELECT'teki ikinci
belirtilen veriyi temsil ediyor.
--GROUPING OPERATION SORU 4--
```

```
-- Write a query that returns BRANDS with an average product price more than
1000
SELECT A.brand_name, AVG(B.list_price) AS AVG_PRICE
FROM production.brands A, production.products B
WHERE A.brand_id = B.brand_id
GROUP BY
                A.brand_name
HAVING AVG(B.list_price) > 1000
ORDER BY
                2 DESC
--GROUPING OPERATION SORU 5--
--  Write a query that returns the net price paid by the customer for each order.
(Don't neglect discounts and quantities)
SELECT *, (quantity * list_price * (1-discount)) as net_price
--list_price-list_price*discount olarak da yazılabilir
FROM sales.order_items
-- bu query ile önce her bire order_id için list_price ile indirim uygulanmış
net_price ları görüyoruz.
-- order'larda birden fazla ürün sipariş verilmiş olduğunu görmüştüm.
-- O yüzden ürünleri order_id olarak gruplandırıp her grup için toplama (SUM)
yaparak
-- her order için toplam net_price'ı görmüş olacağım
SELECT order_id, SUM(quantity * list_price * (1-discount)) as net_price
FROM sales.order_items
GROUP BY
                order_id
--- SUMMARY TABLE---
SELECT *
INTO NEW_TABLE -- INTO SATIRINDAKİ TABLO İSEMİ İLE YENİ BİR TABLO
OLUŞTURUYORUZ.
FROM SOURCE_TABLE  -- FROM'DAN SONRASI KAYNAK TABLOMUZ
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_summary
FROM 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
SELECT *
FROM sales.sales_summary
ORDER BY 1,2,3
```

```
-- Bundan sonra bu tabloyu kullanacağım!
--- GROUPING SETS----
-- 1. Toplam sales miktarını hesaplayınız.
SELECT SUM(total_sales_price)
FROM sales.sales_summary
-- 2. Markaların toplam sales miktarını hesaplayınız.
SELECT Brand, SUM(total_sales_price)
FROM sales.sales_summary
GROUP BY
		Brand
-- 3. Kategori bazında toplam sales miktarını hesaplayınız
SELECT Category, SUM(total_sales_price)
FROM sales.sales_summary
GROUP BY
		Category
-- 4. Marka ve kategori kırılımlarındaki toplam sales miktarlarını hesaplayınız
SELECT Brand, Category, SUM(total_sales_price)
FROM sales.sales_summary
GROUP BY
		Brand, Category
-- BU İŞLERMLERİ GROUPING SETS YÖNTEMİ İLE YAPALIM :---
SELECT brand, category, SUM(total_sales_price)
FROM sales.sales_summary
GROUP BY
		GROUPING SETS(
						(Brand),
						(category),
						(brand, category),
						()	-- boş parantez ile
						)
ORDER BY
		1,2
----- ROLLUP GRUPLAMA-----
SELECT
		d1,
		d2,
		d3,
		aggregate_function
FROM
		table_name
GROUP BY
		ROLLUP (d1,d2,d3);
		-- önce tüm sütuınları alıyor sonra sağdan başlayarak teker teker
silerek her defasında yeniden bir gruplama yapıyor;
		-- önce üç sütuna göre grupluyor, sonra sondakini atıp ilk 2 sütuna
göre grupluyor
```

```
                -- sonra sondakini yine atıp ilk sütuna göre grupluyor
                -- sonra hiç gruplamıyor.--
SELECT brand, category, SUM(total_sales_price)
FROM sales.sales_summary
GROUP BY
                ROLLUP (Brand, Category)
ORDER BY
                1,2
                ;
--- CUBE GRUPLAMA----
--- önce önce üç sütunu birden grupluyor
-- sonra kalanları 2'şer 2'şer 3 defa gruplama yapıyor
-- sonra kalanları teker teker grupluyor
-- en son gruplamıyor.
SELECT brand, category, SUM(total_sales_price)
FROM sales.sales_summary
GROUP BY
                CUBE (Brand, Category)
ORDER BY
                1,2
                ;
```

ROLLUP, en ayrıntılıdan genel toplama kadar ihtiyaç duyulan herhangi bir toplama düzeyinde alt toplamlar oluşturur. CUBE, ROLLUP'a benzer ama tek bir ifadenin tüm olası alt toplam kombinasyonlarını hesaplamasını sağlar.