SQL INCLASS NOTLARI

- Önce **FROM**'u okur. ***Hangi tabloya gideceğini belirler***.
- Sonra **WHERE**'i okur; ***gittiği bu tablodan hangi filtrelemeyi yapacağını belirler.***
- Sonra varsa **GROUP BY** ile ***gruplama*** yapar.
- **Aggragate** fonsiyonlar için **GROUP BY** kullanılıyorsa **filtrelemeyi HAVING** ile yapıyoruz**. HAVING, Aggregate** fonksiyonların filtrelenmesinde kullanılır. (**Tablo içindeki sütunlarda filtreleme yaparken WHERE kullanıyoruz, Aggregate fonksiyonlar için ise HAVING..)**
- **SELECT**'i ise bütün bunlardan sonra okur. Artık ***şekillenen tablodan hangi alanı getirmek istiyorsanız onu SELECT'te belirtirsiniz.***
- **ORDER BY** ve **LIMIT** ise select ile belirlenen çıktının ***hangi sırayla getirileceğini*** ve hangi satırlarının ***limitlenip*** getirileceğini belirliyor.

****

Birden fazla satıra işlem yapan AVG, SUM, COUNT vs. kullanırken önüne bir sütun ismi yazmıyoruz. Çünkü yazarsak o sütunun bir satırına karşılık değer geleceği için tek satırı alır.

Örnek; COUNT tek satır sonuç döndürdüğü için önüne yazacağın sütun için tek satırı alır.

Aggragate fonsiyon kullanırken buna dikkat et. Ama MIN, MAX'ın önüne sütun ismi girebilirsin.

****

Aggregate fonksiyonları **yalnızca bir veri** üretir.

SUM ve **AVG**, yalnızca **integer** değerlere uygulanır.

COUNT *, **NULL** dahil **tüm satırları sayar**.

COUNT (colum_name) şeklinde sütun ismi girilirse **NULL'lar hariç satırları sayar.**

COUNT * hariç diğer hepsi **(min, max vs.) Null ları göz ardı eder.** Zaten **MIN, MAX** vs. de ***** **ile kullanma şekli yok.**

****

SQL, WHERE'i Aggregate fonksiyonundan önce çalıştırır ki önce sınırlayıp, sınırlanmış veriye fonksiyon işlemi uygulansın.

Böylece sistemi yormamış oluyor.GROUP BY da WHERE'den sonra çalıştırılıyor. Aynı nedenden dolayı.

****

Hatırlayın biz EDA yaparken hep verimizi gruplayıp bir aggregate uyguluyorduk. Çünkü genellikle **verimizi kategorilere göre analiz etmek isteriz.**

SQL'de, tıpkı Pandas'ın **groupby()** metodunda olduğu gibi **GROUP BY** ifadesi; aynı kategori value'suna sahip satırları özet satırlarında gruplandırır.

Böylece daha iyi analiz için veriyi ayrı gruplara bölüp hesaplama yapabiliyoruz.

df.groupby("company")[[quantity]].mean()
&
SELECT company AVG(quantity) FROM bla-bla GROUPBY company;

****

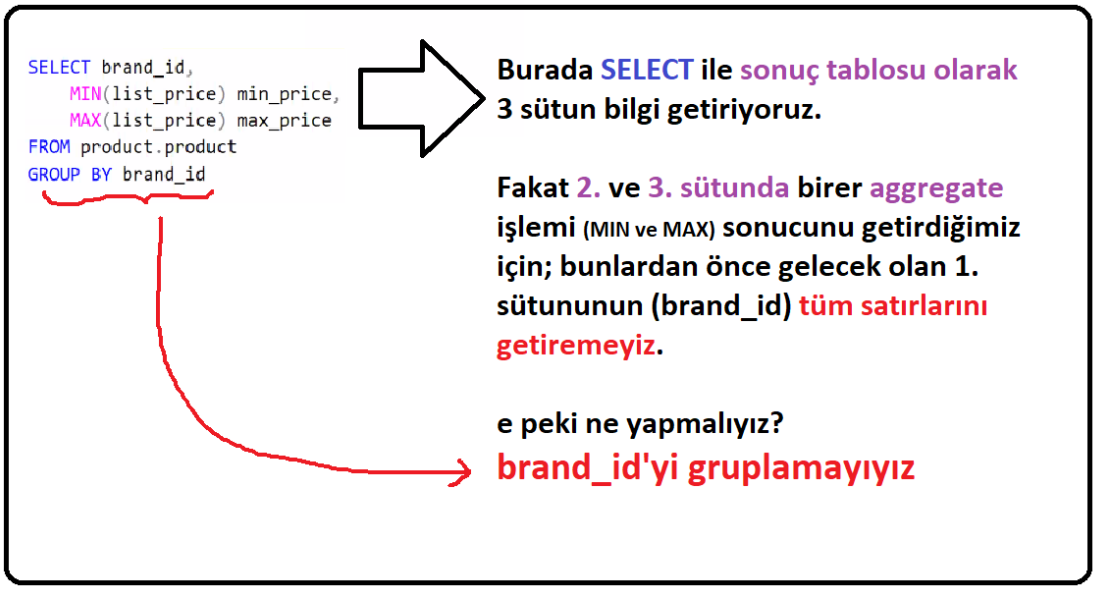
GROUP BY, aggregate function'ı çağırmadan önce sonuçları gruplandırır. Bu, tüm sorgu yerine gruplara aggregate func. uygulamanıza olanak tanır.

- **WHERE cümlesi, aggregate'ten önceki veriler üzerinde operasyon yapar (çalışır).**
- **WHERE cümlesi, GROUP BY cümlesinden önce çalışır. Dolayısıyla ana tablomuzun sadece WHERE cümlesindeki şartları sağlayan satırları gruplanır.**
- **ORDER BY, GROUP BY'dan sonra gelir. (Sonucu sıralar.)**

1. **Yani ilk önce WHERE şartı uygulanıp data filtrelenir,**
2. **sonra üzerine GROUP BY uygulanır,**
3. **sonra aggregate function uygulanır.**
4. **Son olarak da sonuç ORDER BY a göre sıralanır.**

(çalışma sütunlarını + koşula göre seç + fonksiyona göre gruplandır + sıralamasını belirle)

****



****

DİKKAT!!

WHERE ile ana tabloda bir filtreleme yapıyoruz. Ana tablo içinde herhangi bir filtreleme yapmayacaksan WHERE satırı kullanmayacaksın demektir.

ORDER BY, SELECT'ten sonra çalışıyor. Dolayısıyla SELECT'te yazdığım Allias'ı kabul eder!

SELECT satırında yazdığın sütunların hepsi **GROUP BY**'da olması gerekiyor!

ORDER BY satırındaki ilk parametre (örneğin) 2 ise bu SELECT satırındaki 2. sütuna göre sırala demektir.

HAVING ile ise query ile dönen sonuç üzerinde bir filtreleme yapıyoruz.

HAVING ile sadece aggregate sonucuna bir filtre uyguluyoruz. Dolayısıyla HAVING, GROUP BY ile birlikte kullanılıyor.

HAVING'de kullandığın sütun, aggregate te kullandığın sütunla aynı olmalı.

****

SQL KOMUTLARI

SELECT **FROM**	**SELECT** first_name **FROM** employees;
DISTINCT	SELECT DISTINCT column_name FROM table_name SELECT **DISTINCT** first_name, gender FROM employees; ***{Distinct=farklı. Unique=benzersiz. Belirtilen column(lar)daki distinct/ unique* olan datayı çek getir. Birden fazla column belirtmişsek onların kesişimlerini alıyor}**
WHERE	SELECT column_name(s) FROM table_name WHERE condition(s); SELECT * FROM student_table **WHERE** grade > 70 ***{if conditions gibi düşün}***
LIMIT	SELECT column_name(s) FROM table_name LIMIT number_rows; SELECT * FROM student_table WHERE grade > 70 **LIMIT** 2; ***{LIMIT'ten sonraki sayı kadar satır bilgisini seç getir}***
ORDER BY **ASC, DESC**	SELECT column_name(s) FROM table_name ORDER BY column_name(s) ASC|DESC; SELECT * FROM employees **ORDER BY** first_name **ASC**; SELECT first_name, last_name, salary FROM employees **ORDER BY** salary **DESC**; SELECT column_name(s) FROM table_name ORDER BY column1 ASC|DESC, column2 ASC|DESC, columnN ASC|DESC; SELECT * FROM employees **ORDER BY *gender* DESC, *first_name* ASC**;
WHERE + ORDER BY	SELECT column_name(s) FROM table_name WHERE condition ORDER BY column_name(s)s ASC|DESC; SELECT * FROM employees **WHERE** salary > 80000 **{Önce WHERE geliyor sonra ORDER BY!!}** **ORDER BY** first_name DESC;
WHERE AND	WHERE left_conditon AND right_condition SELECT * FROM employees **WHERE** job_title = 'Data Scientist' **AND** gender = 'Male';

WHERE **OR**	WHERE first_condition OR second_condition SELECT * FROM employees **WHERE** job_title **=** 'Data Scientist' **OR** gender **=** 'Male';
WHERE NOT	WHERE NOT first_condition **WHERE NOT** gender = 'Female';
WHERE **BETWEEN** **AND**	WHERE test_expression BETWEEN low_expression AND high_expression Also we could write this query like this: WHERE test_expression >= low_expression AND test_expression <= high_expression SELECT * FROM employees **WHERE** salary **BETWEEN** 80000 **AND** 90000; Also we could write this query like this: SELECT * FROM employees **WHERE** salary **>=** 80000 **AND** salary **<=** 90000; {Between'de sol ve sağdaki **iki değer de DAHİL** olarak kabul edilecek. **İstisnası**, eğer tarih verisi **date-time** şeklinde ise **sondaki tarih dahil olmuyor**. Buna bir gün eklememiz gerekiyor. Çünkü 23:59'dan sonra bir
WHERE **NOT** **BETWEEN** **AND**	WHERE test_expression NOT BETWEEN low_expression AND high_expression SELECT * FROM employees **WHERE** salary **NOT BETWEEN** 80000 **AND** 90000; *Also we could write this query like this:* SELECT * FROM employees **WHERE** salary **<** 80000 **OR** salary **>** 90000;
BETWEEN AND ORDER BY	SELECT * FROM employees **WHERE** hire_date **BETWEEN** '2018-06-01' **AND** '2019-03-31' **ORDER BY** hire_date;

WHERE **NOT IN**	WHERE column_name IN (value_list) SELECT * FROM employees **WHERE** job_title **IN** *('Data Scientist', 'Business Analyst', 'Project Manager', 'Web Developer')*; SELECT * FROM employees **WHERE** job_title **NOT IN** ('Operations Director', 'HR Manager', 'Sales Manager');
WHERE **LİKE** **%** **_**	SELECT column_name(s) FROM table_name WHERE column_1 LIKE pattern; SELECT * FROM student_info **WHERE** county **LIKE** 'Wo**%**'; -- Wo ile başlayan ülkeler SELECT * FROM student_info **WHERE** field **LIKE** '**%**Developer'; --sonu Developer ile bitenler SELECT first_name FROM employees **WHERE** first_name **LIKE** 'El_is'; ***{Linda'daki i ve n harleri için 2 alt tire kullandık}***
PRAGMA	-- if you want to make **LIKE** operator **case-sensitive**, you need to use **PRAGMA** statement PRAGMA case_sensitive_like = true; **PRAGMA** *case_sensitive_like = true*; SELECT * FROM student_info WHERE field LIKE '%developer';,

COUNT( )	SELECT COUNT(column_name) FROM table_name; SELECT **COUNT(first_name)** FROM student_info; ***{COUNT (*) şeklinde kullanırsan tekrarları ve içeriği NULL olan satırları da getirir. Sütun ismiyle kullanırsan tekrarları getirir ama NULL ları getirmez. NULL ların sayılmasını istemiyorsak * kullanmayıp parantez***
AS	SELECT COUNT(first_name) **AS** count_of_students FROM student_info; ***{AS komutu ile listelenen bilgiye takma ad veriyoruz böylece ekrana getirilen sonucun ne olduğunu daha iyi anlayabiliyoruz.}***
COUNT(DISTINCT)	SELECT **COUNT(DISTINCT field)** AS count_of_field FROM student_info; **{Birbirinden farklı (unique) kaç bilgi varsa onun sayısını verir.}**
COUNT WHERE	SELECT COUNT(*) AS count_of_students FROM student_info WHERE state = 'Virginia';
MIN	SELECT MIN(column_name) FROM table_name; ***{ignores the NULL values}*** SELECT **MIN(salary)** AS lowest_salary FROM employees; SELECT **MIN(salary) AS** lowest_salary FROM employees **WHERE** gender = 'Female'; SELECT **MIN(hire_date)** AS earliest_date FROM employees;

MAX	SELECT MAX(column_name) FROM table_name; ***{ignores the NULL values}*** SELECT **MAX(salary)** AS highest_salary FROM employees; SELECT **MAX(salary)** AS highest_salary FROM employees **WHERE** gender = 'Male'; SELECT **MAX(hire_date)** AS newest_date FROM employees;

SUM	SELECT SUM(column_name) FROM table_name; {***returns the sum of a numeric column}*** ***Sadece numeric datalara uygulanıyor!!*** SELECT **SUM(salary)** AS total_salary
AVG	SELECT AVG(column_name) FROM table_name; ***{calculates the average of numeric colum}*** ***Sadece numeric datalara uygulanıyor!!*** SELECT **AVG(salary)** AS average_salary FROM employees; SELECT name, AVG(Milliseconds) FROM tracks WHERE Milliseconds > 393599.212103911; SELECT name FROM tracks WHERE Milliseconds > (SELECT AVG(Milliseconds)

GROUP BY	SELECT column_1, **aggregate_function**(column_2) FROM table_name {**Her zaman WHERE'den sonra gelir** GROUP BY column_1; **Her zaman ORDER BY'dan önce gelir.}** **{veri grubu başına yalınızca bir sonuç döndürür. Bunun için verileri unique olarak alır yani kendi içinde DISTINCT yapar. Mesela aşağıdaki örneğe göre Male ve Female'ler için iki ayrı sonuç döndürür.}** !!! ***SELECT'teki nonaggregate ifade (column_1), GROUP BY cümlesinde olmalı!!! Yani neyi GROUP BY'ın yanına yazıp grupluyor isek aynen onu SELECT'in yanına yapıştır.*** SELECT *gender*, **COUNT**(gender) FROM employees **GROUP BY** *gender*; ***{gender'ların sayılarını gender lara göre***

GROUP BY ÖRNEKLERİ	SELECT gender, **COUNT**(job_title) FROM employees **WHERE** job_title = 'Data Scientist' **GROUP BY** gender; **{job_title'lar arasında Data Scientist'leri çek, onları gender lara göre gruplandır.}** SELECT job_title, count(job_title) FROM employees GROUP BY gender, job_title; {gender'lara göre hangi iş pozisyonlarından kaçar tane var} SELECT job_title, gender, count(gender) FROM employees GROUP BY job_title, gender ; { iş pozisyonlarına göre gender'lardan kaçar tane var} SELECT job_title, gender, count(gender) FROM employees WHERE job_title='Data Scientist' {Data Scientist olarak gender'lardan kaçar tane GROUP BY job_title, gender ; var} SELECT Composer, COUNT(TrackId) FROM tracks GROUP BY Composer; {Her bestecinin kaçar şarkısı var? Burada null olanları da alır. SELECT Composer, COUNT(TrackId) FROM tracks WHERE Composer IS NOT NULL {Her bestecinin kaçar şarkısı var? GROUP BY Composer; burada null olanları çıkarttık SELECT Country, COUNT(CustomerId) FROM customers GROUP BY Country; {her ülkeden kaçar tane müşteri var?}

	GROUP BY, Aggregate function'ı çağırmadan önce sonuçları gruplandırır. Bu, tüm sorgu yerine gruplara aggregate func. uygulamanıza olanak tanır. **WHERE cümlesi, aggregate'ten önceki veriler üzerinde operasyon yapar (çalışır).** **WHERE cümlesi, GROUP BY cümlesinden önce çalışır.** **Dolayısıyla ana tablomuzun sadece WHERE cümlesindeki şartları sağlayan satırları gruplanır.** **ORDER BY, GROUP BY'dan sonra gelir.** ***Yani ilk önce WHERE şartı uygulanıp data filtrelenir, sonra üzerine GROUP BY, sonra aggregate function uygulanır. Son olarak da sonuc***
MIN **GROUP BY**	SELECT gender, **MIN**(salary) AS min_salary FROM employees **GROUP BY** gender;
MAX **GROUP BY**	SELECT gender, **MAX**(salary) AS max_salary FROM employees **GROUP BY** gender;
GROUP BY **ORDER BY**	SELECT gender, MAX(salary) AS max_salary FROM employees **GROUP BY** gender **ORDER BY** max_salary DESC ; {salary içerisindeki maksimum ücreti genderlarla grupla yani F'lerden en yüksek alan ile M'lerden en yüksek alanları bir araya getir. bunları yüksekten alçağa doğru sırala} ***{ yukarda MAX(salary)'e max_salary olarak takma ad vermiştik. Ve bu adı ORDER BY max_salary ifadesinde kullandık. Bunun yerine ORDER BY MAX(salary) olarak da kullanabiliyoruz.}***
SUM / AVG **GROUP BY**	SELECT gender, **SUM(salary) AS** total_salary FROM employees **GROUP BY** gender; SELECT gender, **AVG(salary) AS** average_salary FROM employees GROUP BY gender;