**SQL'İN KENDİ İÇİNDEKİ İŞLEM SIRASI:**

**FROM** : hangi tablolara gitmem gerekiyor?

**WHERE** : o tablolardan hangi verileri çekmem gerekiyor?

**GROUP BY** : bu bilgileri ne şekilde gruplayayım?

**SELECT** : neleri getireyim ve hangi aggragate işlemine tabi tutayım.

**HAVING** : yukardaki sorgu sonucu çıkan tablo üzerinden nasıl bir filtreleme yapayım (mesela list\_price>1000)

* + - Gruplama yaptığımız aynı sorgu içinde bir filtreleme yapmak istiyorsak HAVING kullanacağız
    - HAVING kullanmadan; HAVING'ten yukarısını alıp başka bir SELECT sorgusunda WHERE şartı ile de bu filtrelemeyi yapabiliriz.

**ORDER BY** : Çıkan sonucu hangi sıralama ile getireyim?

**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!

**CROSS JOIN:**

* Tam bir join gibi değildir. İki verinin kombinasyonundan bahsediyorsak Cross Join kullanıyoruz.
* Cross Join A tablosundaki her (gözlemi) satırı B tablosundaki her bir satırla eşleştiriyor.
* Cross Join edeceğimiz iki tablo arasında ortak sütun olması gerekmiyor.

**SELF JOIN**: INNER JOIN gibidir. Aynı tabloyu tekrar kendisi ile join ederek kendi içerisinde bir bilgiye ulaşmaya çalışıyoruz.

JOIN'i **FROM-WHERE** ile de yapabiliyoruz. Bunun için birleştireceğimiz tabloları FROM satırında VİRGÜLLE yan yana yazıp, ON satırındaki eşleşmeyi WHERE satırında yapıyoruz.

Soruda average veya toplam gibi aggregate işlemi gerektirecek bir istek varsa hemen "GROUP BY" kullanmam gerektiğini anlamalıyım.

Bir sayma işlemi, bir gruplandırma bir aggregate işlemi yapıyorsan isimle değil ID ile yap. ID'ler her zaman birer defa geçer (Unique’tir), isimler tekrar edebilir (ör: bir kaç product'a aynı isim verilmiş olabilir)

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

HAVING ile sadece aggregate ettiğimiz sütuna koşul verebiliriz!

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

SQLite HAVING satırında takma ad kullanmana izin veriyor ancak SQL Server izin vermiyor!

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

**VIEWS**

* Subquery'ler, CTE(Common Table Expression)'lar, VIEW'lar hep aynı amaca hizmet ediyor. Tablolarla daha rahat çalışmamızı sağlıyorlar.
* Diğer bir avantajı da performansı artırmaktır. Siz query'nizi joinlerle tek bir query içinde değil, subery lerle, CTE'lerle,
* VIEW'larla daralta daralta (daraltılmış tablolarla) sonuca gitmeye çalışıyorsunuz.
* VIEW : Tek bir tabloda yapacağımız işlemleri aşamalara bölerek yapmamızı sağlıyor. Bu da hızımızı arttırıyor.
* VIEW ile aynı tablo gibi oluşturuyoruz ve bu VIEW'a kimleri erişebileceğini belirleyebiliyoruz. bu da security sağlıyor.
* VIEW'ların kullanımı da oluşturması basittir. büyük tablonun içerisinde biz bir kısım ilgilendiğimiz verileri alıp onlar üzerinden çalışıyoruz.
* Ana tablo güncellendikçe VIEW' da otomatik olarak güncellenir.
* VIEW'lar çok az yer kaplar. Çünkü asıl tablonun bir görüntüsüdür.

**AVANTAJLARI:**

 Performans + Simplicity + Security + Storage (edited)

Az önceki **Self Join** sorusu ile ilgili:

* Hem staff'ler hem manager'ler aynı sales.staffs tablosu içindeler.
* Bu tablo kendi kendine ilişki içerisinde. bu tabloda iki tane sütun birbiri ile aynı bilgiyi taşıyor.
* Burda staff\_id ile manager\_id birbiri ile ilişki içinde. her staff'in bir manageri var ve bu manager aynı zamanda bir staff..
* Mesela staff\_id si 2 olan Charles'ın manager\_id'si 1,  yani staff\_id'si 1 olan kişi Charles'ın manageri demektir.