**Bir tabloda meydana gelen sonucu başka bir tablo veya işlem için kullanmak için 3 yöntem:**

**Subqueries**

**Views**

**Common Table Expression (CTE's)**

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.

**SUBQUERY**'ler:

SELECT, FROM ve WHERE satırlarında kullanılabiliyor.

WHERE'de subquery sonucunda dönen ifadelere göre ana tablo üzerinden bir filtreleme yapacağın anlamına geliyor.

WHERE'in her zaman ana tablo üzerinde filtreleme yaptığını unutma!

SELECT'te subquery içindeki değeri SELECT satırında döndürmek için kullanılıyor.

SELECT satırındaki subquery TEK BİR SÜTUN VEYA SATIR DÖNDÜRMEK ZORUNDA! (sadece bir değer döndürmeli)

FROM da subquery bir tablo getirmesi lazım. Başka kıstaslara göre bir tablo oluşturuyor ve bunu FROM’da kullanmak üzere getiriyor.

**SUBQUERY ÇEŞİTLERİ:**

Single-row : Tek bir satır döndürür. SELECT'te kullanılan gibi.

Multiple-row: Birden fazla değer döndüren subquery

Correlated : üstteki sorgu ile alltaki sorgunun birbiri ile eşlenerek bağlantı kurulduğu subquery

**SINGLE-ROW SUBQUERY**

=, <, >, >=, <=, <>, != operatörleri ile özellikle WHERE satırında kullanılan subquerylerdir.

**MULTIPLE ROW SUBQUERIES**

Birden çok değer döndüren subquerylerdir.

Birçok değer içerisinden bir değer arıyor ve onlar içerisinde bir filtreleme yapacaksam

IN, NOT IN, ANY ve ALL operatörlerini kullanıyorum.\*/

**PİVOT TABLE:**Pivot, satır bazlı analiz sonucunu sütun bazına dönüştürülmesini sağlıyor.GROUP BY gibi bir gruplama yapıyor. Dolayısıyla group by kullanmıyoruz, pivota özel bir syntax kullanıyoruz.Bu syntax içerisinde aggregate işlemi yapıp ilgili sütunlardaki kategorilere göre bir pivot table oluşturuyor.ve o sütunun satırlarını oluşturan her bir kategoriyi birer sütuna dönüştürüyor.Yani satırlardaki value'lar sütunlarda sergileniyor.Pivot tablosunda sütun ve value olarak gözükmesini istediğim sütunları (feature'ları) Pivot'un üstündeki SELECT satırına ekliyorum.Bunlardan VALUE olacak olan sütununa Pivot ile başlayan kod bloğunda AGGRAGATE işlemi uyguluyorum.Unutmayalım ki pivot table, group by işleminin aynısını yapıyor. Aggregate işlemi de oradan geliyor.!! Eğer kaynak tablomdaki bir sütun hem value'lar için kullanılacak (aggregate yapılacak) hem de ayrı bir boyut olarak kullanılacak ise;  
       SELECT satırında bu sütun iki kere yazılmalı (biri ilave boyut için diğeri value'ları oluşturmak için)  
 Fakat "The column 'xxxxxx' was specified multiple times" hatası almamak için birine "Alias" (takma ad) verilmeli!!ÖRNEK:-- önce kaynak tabloyu belirtiyoruz. kaynak tablom bu query:  
SELECT Category, total\_sales\_price  
FROM sales.sales\_summary-- şimdi Pivot işlemi sonucunda ortaya çıkacak tablo için bir SELECT işlemi daha yapmam gerekiyor.  
SELECT \*  
FROM (  
    SELECT Category, total\_sales\_price -- bu parantezin içi kaynak tablom.  
    FROM sales.sales\_summary  
     ) A  
PIVOT  
(  
    SUM(total\_sales\_price)  
    FOR category   -- pivot sütunumuz category, ve value'larımız bu sütundaki bisiklet modelleri olacak. Pivot için belirlediğimiz column'un, grupladığımız column olduğuna dikkat et.  
    IN(  
    [Children Bicycles],  -- category sütunu altında bu kategoriler vardı.  
   [Comfort Bicycles],   -- bunlar pivot table ın sütunları olacak.  
   [Cruisers Bicycles],  
   [Cyclocross Bicycles],  
   [Electric Bikes],  
   [Mountain Bikes],  
   [Road Bikes])  
    ) AS PIVOT\_TABLE -- pivot table'a bir isim vermemiz gerekiyor.

**VIEW:**

AVANTAJLARI: **Performans + Simplicity + Security + Storage**

Tek bir tabloda yapacağımız işlemleri aşamalar 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.

VIEW'lar çok az yer kaplar. çünkü asıl tablonun bir görüntüsüdür.

Ana tablo güncellendikçe VIEW' da otomatik olarak güncellenir.

View yerine bir tablo create edildiğinde ana tablodan verileri çekip ekstradan kaydetmiş olacaktım. Ve ana tablodaki değerler güncellendiğinde bu tablo güncellenmemiş olacaktı. Yani tablo create etmek maliyetli bir işlemdir.

(Müşteri adı soyadı, order\_date, product\_name, model\_year, quantity, list\_price, final\_price (indirimli fiyat)

Bu bilgileri farklı tablolardan alabiliriz. Farklı tablolardan her seferinde aynı sorguyu çalıştırıp bir sonuç almaktansa; bunları ben bir kere kaydedeyim ve tablo güncellendikçe bunlar da güncellensin dediğimde VIEW kullanıyorum.)

**COMMON TABLE ESPRESSIONS (CTE)**

CTE, Subquery mantığı ile aynı. Subquery'de içerde bir tablo ile ilgileniyorduk CTE'de yukarda yazıyoruz.

Başka bir SELECT, INSERT, DELETE veya UPDATE deyiminde başvurabileceğiniz veya içinde kullanabileceğiniz geçici bir sonuç kümesidir.

Başka bir SQL sorgusu içinde tanımlayabileceğiniz bir sorgudur. Bu nedenle, diğer sorgular CTE'yi bir tablo gibi kullanabilir.

CTE, daha büyük bir sorguda kullanılmak üzere yardımcı ifadeler yazmamızı sağlar.

**EXISTS & NOT EXISTS:**

EXISTS komutu, belirtilen bir alt sorguda herhangi bir veri **varlığını test etmek**için kullanılır.

WHERE bloğunda kullanmış olduğumuz IN ifadesinin kullanımına benzer olarak, EXISTS ve NOT EXISTS ifadeleri de **alt sorgudan getirilen değerlerin içerisinde bir değerin olması veya olmaması durumunda işlem yapılmasını sağlar.**

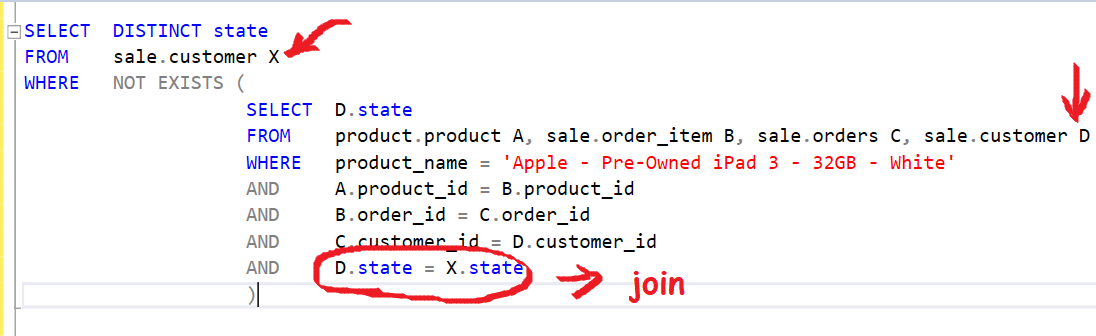
EXISTS: alt sorguda istenilen şartların yerine getirildiği durumlarda ilgili kaydın listelenmesini sağlar.

NOT EXITS : EXISTS‘in tam tersi olarak alt sorguda istenilen şartların sağlanmadığı durumlarda ilgili kaydın listelenmesini sağlar.

**EXIST**kullandığın zaman; subquery herhangi bir **sonuç döndürürse** üstteki query'i ÇALIŞTIR anlamına geliyor.

**NOT EXIST**; subquery herhangi bir sonuç döndürürse üstteki query'i ÇALIŞTIRMA anlamına geliyor.

IN & NOT IN'den farklı olarak EXIST & NOT EXIST'te **query ile subquery'i join etmemiz gerekiyor.**



SORU : Write a query that returns State where 'Trek Remedy 9.8 - 2017' product is not ordered.

(TR) 'Trek Remedy 9.8 - 2017' ürününün sipariş edilmediği State'leri getir.

**NOT IN İLE ÇÖZÜM:**

SELECT DISTINCT state

FROM sale.customer

WHERE state NOT IN (

SELECT DISTINCT D.STATE -- BURADA DISTINCT'e gerek yok.

FROM product.product A, sale.order\_item B, sale.orders C,   
sale.customer D

WHERE A.product\_id = B.product\_id

and B.order\_id = C.order\_id

and C.customer\_id = D.customer\_id

and A.product\_name = 'Trek Remedy 9.8 - 2017')

NOT IN ile bu product'ın siparişinin verildiği (çünkü state'i sales.customers tablosundan çekiyoruz) state'lerin dışında kalan state'leri getir demiş olduk.

**NOT EXIST İLE YAPARSAK:**

SELECT DISTINCT state

FROM sale.customer X

WHERE NOT EXISTS (

SELECT DISTINCT D.STATE -- BURAYA HERHANGİ BİR RAKAM KOYABİLİRSİN. NOT EXIST, SELECT SATIRINA BAKMIYOR!!

FROM product.product A, sale.order\_item B, sale.orders C, sale.customer D

WHERE A.product\_id = B.product\_id

and B.order\_id = C.order\_id

and C.customer\_id = D.customer\_id

and A.product\_name = 'Trek Remedy 9.8 - 2017'

and X.state = D.state

)

**customer**tablosundan **state**'leri getiren bir sorgumuz var fakat gelecek olan state'ler **WHERE kısmındaki koşula göre listelenecek.**

Buna göre: NOT EXIST kullandığımız için; **product\_name** değeri 'Apple - Pre-Owned iPad 3...' **DEĞİL İSE** ilgilli **state**listelenecektir.

**GROUPING SETS :**

**BİRDEN FAZLA GRUPLAMA KOMBİNASYONU YAPMAMA İMKAN SAĞLIYOR.**

GROUP BY'dan farkı : GROUP BY sadece bir sütunun verilerini grupluyordu.

GROUPING SET ile ise birden fazla gruplama varyasyonu yapabiliyoruz.

GROUPING SETS SATIRINDAKİ SIRALAMA ÖNEMLİ DEĞİL, BURADA SETLERİ PARANTEZ İÇİNDE GÖSTERMEK YETERLİ.

SQL, SONUÇ TABLOSUNU "SELECT" SATIRINDAKİ SIRALAMAYA GÖRE OLUŞTURUYOR.

BU YÜZDEN BU SATIRA HANGİ SÜTUNUN ÖNCE YAZILDIĞI ÖNEM ARZ EDİYOR.

ÇÜNKÜ İLK YAZILAN SÜTUNUN ELEMANLARI RESULT TABLOSUNDA TEK TEK SIRA İLE İKİNCİ SÜTUNUN GRUPLANDIRMASINA TABİ OLUYOR!

**ROLLUP GRUPLAMA:**

önce tüm sütunları alıp grupluyor, sonra sağdan başlayarak teker teker sütun 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.

**CUBE GRUPLAMA:**

önce önce üç sütunu birden grupluyor. (d1,d2,d3)

sonra kalanları 2'şer 2'şer 3 defa gruplama yapıyor.

(d1,d2) + (d1,d3) + (d2,d3)

sonra kalanları teker teker grupluyor. (d1) + (d2) + (d3)

en son gruplamıyor. ()