**IN CLASS DERSLERIN CLASS CHAT’INDEN NOTLAR**

**Session-7**

* Arkadaşlar biz nerdeyiz, ne yapıyoruz, şu an neyle meşgulüz diye soruyorsanız:  
  **DDL** : **Data Definition Language**: Veri girişi yapılmadan önce database'i oluşturmak ve configüre etmek için kullandığımız komutlardan oluşur (**CREATE, ALTER, DROP**)**DML** : **Data Manipulation Language** : Dataya ulaşmak ve manipule etmek için kullanılan komutlardan oluşur (**INSERT, UPDATE, DELETE, SELECT**)Biz bu derste ve uzunca bir süredir hep bir DML olan **SELECT statement**'lar kullandık. **Yani sorgu yaptık.**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

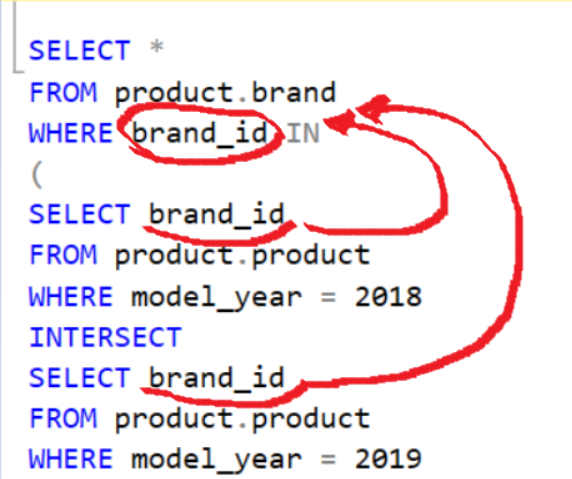
* yalnızca select ile sorgu sonucu getirdik. database de değişiklik yapan bir DDL command kullanmadık.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

* Arkadaşlar bu arada burada order\_status'u WHEN'in içine (sağına) alırsak eşittir (=) kullanmamız gerekir.  
  SELECT order\_status,  
  CASE order\_status WHEN 1 THEN 'Pending'  
  ile  
  SELECT order\_status,  
  CASE WHEN order\_status = 1 THEN 'Pending'  
  aynı işi görür.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

* WHERE clause da hangi alana(column a) condition getiriyorsak SUBQUERY(ler)in SELECT'inde de **aynı alanı getirmek zorundayız**. ***(elmalarla elmaları kıyaslamak için)***



* UNION, select ile getirdiğimi sonuçtaki aynı değerleri tekrarsız getirdiği için, eğer querymizi bu şekilde;  
  select brand\_id  
  from product.brandunionselect category\_id  
  from product.category  
  bazı satırları aynı değerler olacak şekilde **brand\_id**ve **category\_id**leri getirecek olursak bakın burda 40 satır değer döndürecek.

**fakat output ta farklı değerler gelecek şekilde:**  
select brand\_name  
from product.brandunionselect category\_name  
from product.category  
**brand\_name** ve **category\_name** i seçersek, o zaman 56 sonuç geldiğini göreceksiniz.

SELECT first\_name, last\_name

FROM sale.customer

WHERE first\_name = 'Thomas'

UNION ALL

SELECT first\_name, last\_name

FROM sale.customer

WHERE last\_name = 'Thomas'

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

ismi-soyismi  **Thomas Thomas**olan olsaydı UNION ALL durumunda sizce output ne olurdu arkadaşlar

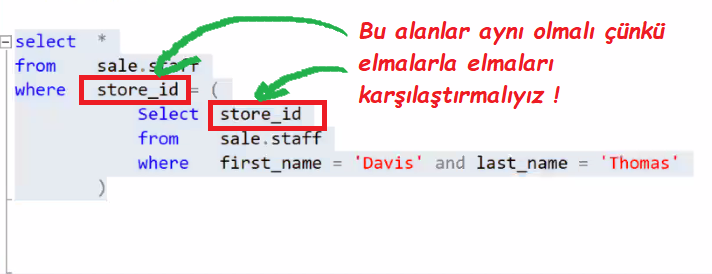
* [**D1528-İlhan**](https://app.slack.com/team/U02RZNNV2GP)

unıon all da iki değer union da tek değer gelirdi

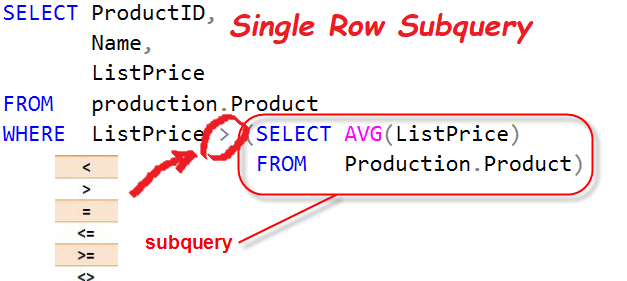
**Session-8**

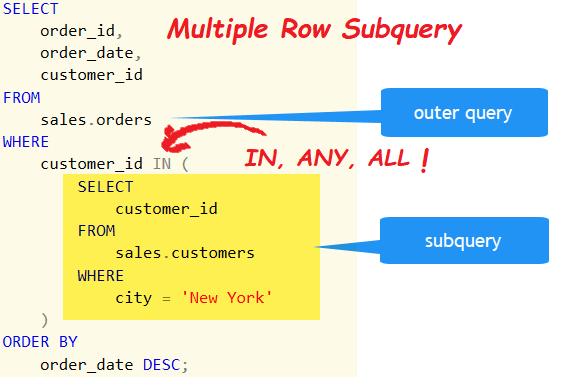
***Join* ile yapabileceğimiz işi *Subquery*ile de yapabiliyor isek*****subquery***'i **tercih etmeliyiz** çünkü daha az maliyetli, sistemi join kadar yormaz ve daha hızlıdır.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***



**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***





**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

* **Kıymetli hocamın soruyu çözerkenki izlediği yola ve adımlarına dikkat edelim.**  
  1. Soru benden ne istiyor?

2. İstenen veri hangi tablolarda yer alıyor (join etmem gerekiyor mu)?

3. Soruda bir şart/filtreleme var mı(WHERE clause)?

4. Gruplama var mı(GROUP BY)?

5. Sıralamam gerekiyor mu (ORDER BY)

6. Subquery kullanacak mıyım?

Acele etmeden adım adım gidiyor. Önce subquery'i yazıyor. Ordan sonucu alıp o sonucu nerde kullanacaksa oraya yapıştırıyor.  Sonra üst sorguyu tamamlıyor.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

* Categorical sütunlarda Max kullanmak yerine ALL yöntemi daha kullanışlı olacaktır.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**ANY operator:**

* Boolean değer döndürür.
* Eğer subquery'nin değerlerinden herhangi biri koşulu karşılıyorsa TRUE döndürür. ***Yani aralıktaki değerlerden herhangi biri TRUE ise koşulun TRUE olacağı anlamına gelir.***

**ALL operator:**

* Boolean değer döndürür.
* Ancak tüm subquery değerleri koşulu sağlıyorsa TRUE döndürür. ***Yani aralıktaki tüm değerler için işlem TRUE ise şart o zaman TRUE olacaktır.***
* SELECT, WHERE ve HAVING statement'ları ile birlikte kullanılır.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

* **EXIST ve NOT EXIST ne yapar? arasındaki fark nedir?**  
    
  **EXISTS**operatörü, bir alt sorgudaki kayıtların varlığını test eder, ***eğer alt sorgudan kayıt dönüyorsa TRUE değerini döndürür. kayıt yoksa FALSE döndürür.*NOT EXISTS** operatörü de alt sorgudaki kayıtların varlığını test eder, eğer alt sorgudan kayıt dönmüyorsa TRUE döndürür. Kayıt varsa FALSE döndürür.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

* **CTE‘ler (Common Table Expressions) nedir ?**  
  CTE (Common Table Expression), **geçici olarak var olan** ve genelde **yinelemeli(recursive) ve büyük sorgu ifadelerinde kullanım için**olan bir sorgunun sonuç kümesi olarak düşünebiliriz.
* Veritabanı görünümleri(**views**) ne benzetebiliriz, ancak hiç bir şekilde alanların (**field&column**) deklare edilmesi gerekmez. CTE’lerin sonuçları depolanmaz ve **yalnızca işlem süresince var olur**.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\* CTE’leri nasıl oluştururuz ?**

* “WITH” keyword’ü ile başlatın.
* Tablo olarak kullanacağımız, geçici bir “isim” ataması yapın.
* İsim ataması yaptıktan sonra, “AS” ile devam edin.
* İsteğe bağlı olarak, field’ların isimlerini yazabilirsiniz (field1,field2)
* Sonuç için bir sorgu yazın.
* Birden fazla CTE’yi bir araya getirmek için, her CTE’den sonra “,” ekleyerek 2–4. adımları tekrarlayın.
* “CTE” dizilimi bittikten sonra, CTE’den referans alacak şekilde sorgunuzu yazın.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

## CTE’ler bize hangi konularda yardımcı olur ?

**View**’ler, **Temp Table**’lar ve **Tablo Tipi Değişkenler**gibi kullanıcıların okunabilirlik deneyimlerini arttırma, soyutlaştırma, kolay yazılabilirliği ve en zor sorguların bile basit bloklarına dönüşmesine yardımcı olurlar. Bunun dışında ise, recursive sorgularımız için vaz geçilmezlerdir.

## Peki CTE’leri nasıl oluştururuz ?

* “**WITH**” keyword’ü ile başlatın.
* Tablo olarak kullanacağımız, geçici bir “**isim**”ataması yapın.
* İsim ataması yaptıktan sonra, “**AS**” ile devam edin.
* İsteğe bağlı olarak, **field**’ların isimlerini yazabilirsiniz (field1,field2)
* Sonuç için bir sorgu yazın.
* Birden fazla **CTE’**yi bir araya getirmek için, her **CTE**’den sonra “**,**” ekleyerek 2–4. adımları tekrarlayın.
* “**CTE**” dizilimi bittikten sonra, **CTE’**den referans alacak şekilde sorgunuzu yazın.

## Syntax

WITH expression\_name1 AS(

CTE query definition 1

)[, expression\_name\_X AS (

CTE query definition X

)[ ,...n ]]

SELECT expression\_A, expression\_B, ...

FROM expression\_name\_X

## Kod tekrarını önleyerek nasıl basit bir sorgu hazırlarız ?

Normal şartlarda **CTE** kullanmasaydık sorgu içerisinde kullandığımız Scalar ve System fonksiyonlarını **GROUP BY**, **ORDER BY** , **HAVING** gibi **Keyword’**ler ile de kullanmak zorunda kalırdık.

## Basit bi örnek ile devam edelim.

Aşağıda ki sorguda, normal şartlarda “**Yıl**” kısmında kullandığımız **DATEPART** fonksiyonunu, **GROUP BY** işleminde de kullanmamız gerekecekti. **CTE**yardımı ile kod tekrarını önlemiş ve ham sorgumuzu düzenlenebilir olarak hala saklayabiliyoruz.

DB : Northwind

|  |
| --- |
| WITH CTE AS ( |
|  | SELECT DATEPART(YEAR,OrderDate) as [Year], |
|  | [Order Details].UnitPrice, |
|  | [Order Details].Quantity |
|  | FROM Orders |
|  | LEFT JOIN [Order Details] ON [Order Details].OrderID = Orders.OrderID |
|  | ) |
|  | SELECT [Year], |
|  | SUM(UnitPrice \* Quantity) as TotalPrice |
|  | FROM CTE |
|  | GROUP BY [Year] |

## Recursive kullanımı için de bir örnek yapalım.

Başlangıç ve bitiş tarihi belirleyelim, sorgunun sonucunda ise bize her güne ait tarih verisi aynı zamanda kaçıncı gün olduğu bilgisini sunsun.

|  |
| --- |
| DECLARE @StartDate date = '2021-04-22', |
|  | @EndDate date = '2021-05-22'; |
|  |  |
|  | WITH CTE AS( |
|  | SELECT DATE = @StartDate, DAY = 1 |
|  | UNION ALL |
|  | SELECT DATE = DATEADD(DAY, 1, DATE), DAY = DAY+1 |
|  | FROM CTE |
|  | WHERE DATE < @EndDate |
|  | ) |
|  | SELECT DATE , DAY |
|  | FROM CTE |
|  | WHERE @StartDate IS NOT NULL |
|  | AND @EndDate IS NOT NULL |
|  | OPTION (maxrecursion 0) |

**Session-9**

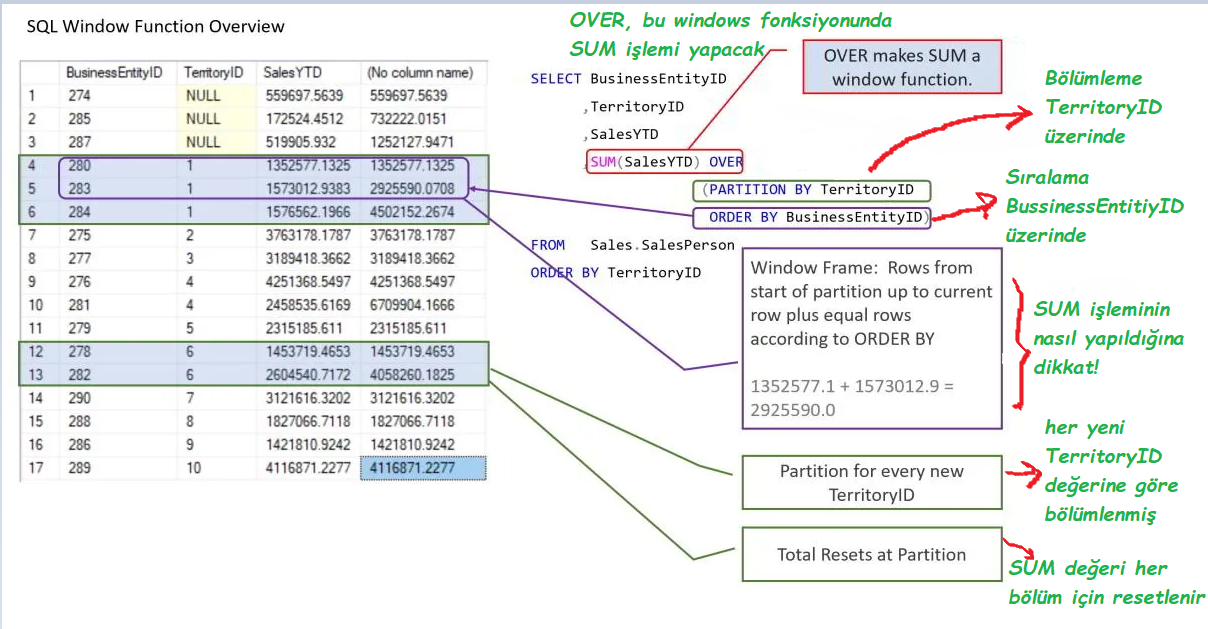
**Nedir bu Windows Function ne işe yarar?**:

Window fonksiyonları sql sorgusu ile elde edilen ***sonuç setini***her fonksiyonun kendi karakterine göre ***parçalara ayırarak*** yine **bu parçalara** kendi fonksiyonlarına göre **değer üretirler.**

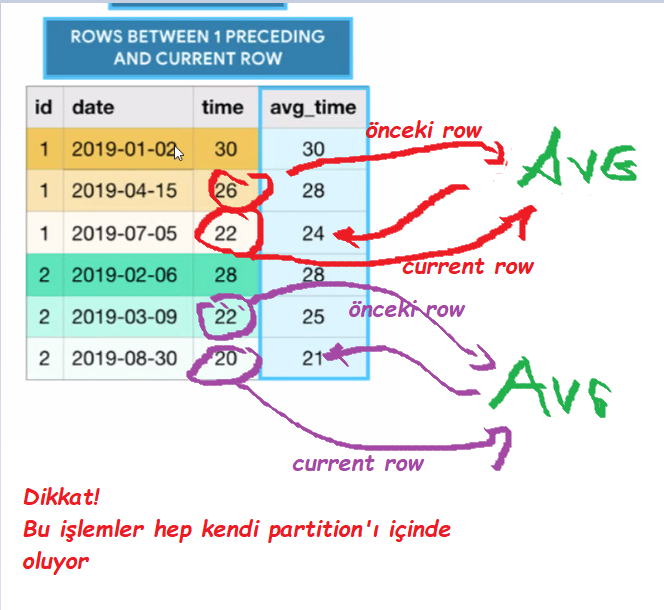
* Bu değerler **SELECT** listesinde veya **ORDER BY** sıralama kriterleri içinde kullanılabilirler.
* Window fonksiyonları kullanılırken **OVER** anahtarı ile kayıt setinin**parçalara bölünmesi sağlanır.**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**WINDOW FUNCTION GENEL BAKIŞ:**

****

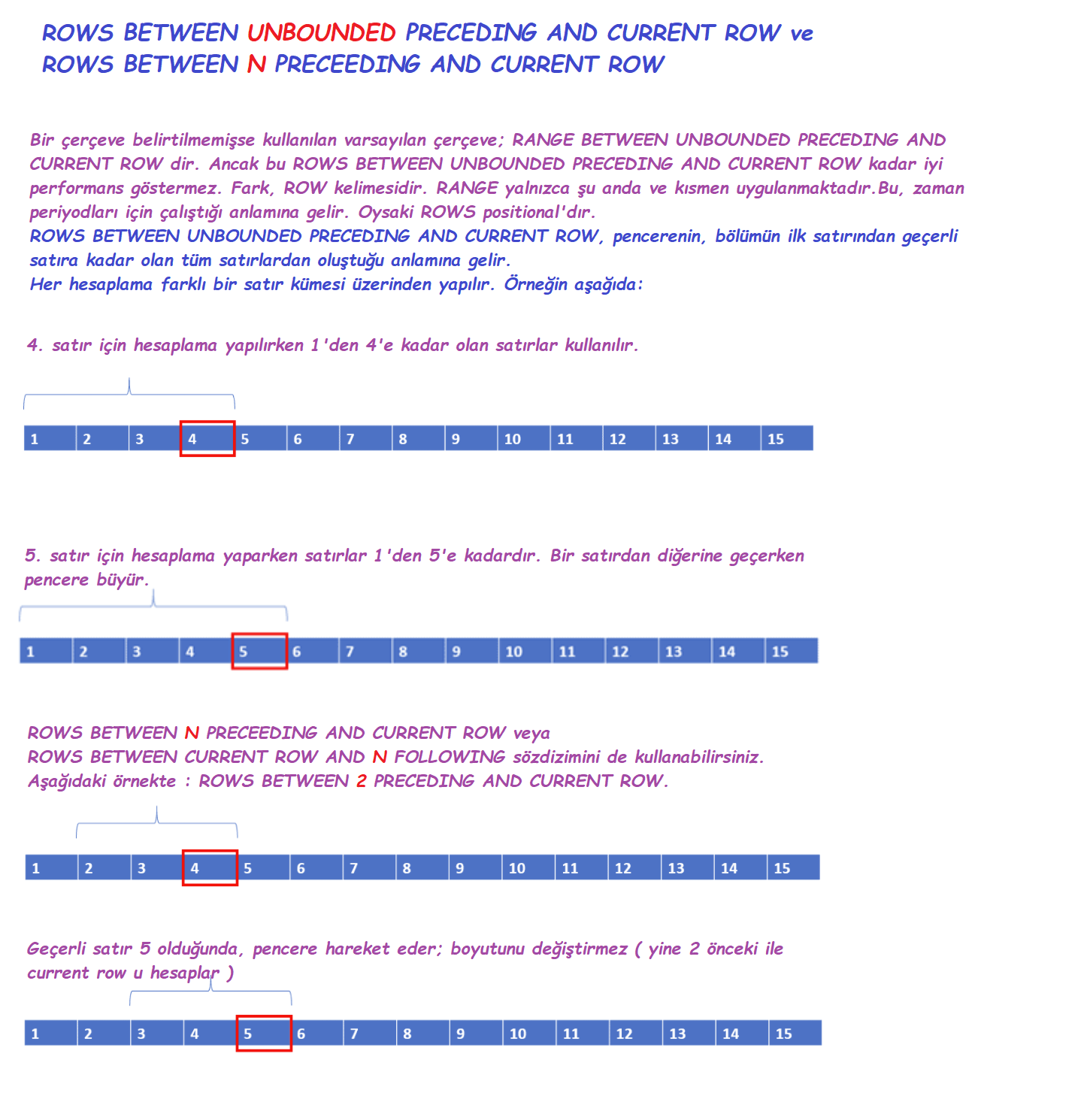
**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

****

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

* DEFAULT: **rows between *unbounded preceeding* and *current row***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

****

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

* order by içinde ***yalnızca order\_date yazdığımızda tarihe göre sıralar.***
* order by içinde order\_date'in yanına bir alan daha yazmış olsak aynı tarihler arasında sıralamayı o alana göre yapar.
* Eğer ***yalnızca order\_date yazarsak ve kayıtta iki aynı tarih olursa***tablo oluşturulurken sıra nasılsa o sıraya göre sıralar.
* **LAG,**current row'dan*belirtilen argümandaki rakam kadar* **önceki** değeri getiriyor.
* **LEAD**, current row'dan *belirtilen argümandaki rakam kadar* **sonraki** değeri getiriyor.

Genellikle **LEAD**VE **LAG**fonksiyonları**SIRALANMIŞ BİR LİSTEYE UYGULANIR !**

O yüzden **ORDER BY KULLANILMALIDIR**!!