

## DELPHİDE SQL KULLANIMI

### Database İlişkisi

ID AD MAAS } Kolon adı (field)

1 Kazım 10000 }

2 Metin 75000 } Kayıt (rows)

... ..... }

Kolon Kolon Kolon

**Table** : Database'de saklanan kolonların birleşiminden oluşan kümedir. Table'ın data tipi yoktur.

**Row** : Tek kayıt demektir.

**Column** : Table'daki kolon adına ait kayıtlardır. Örneğin, 'AD' kolonu demek 'AD' kolonuna girilen verilerin tümüdür.

**Field** : Kolon başlıkları ile kayıt başlığı olanlardır.

**Primary Key** : Unique + not null (Tek olmalı ve boş geçilemez.)

**Foreign Key** : Primary key gibidir. Fakat null değerler alabilir.

Table'lar ilişkisel veri tabanı(compact) olmalıdır. Tablolar arasında bir ilişki kurulmalıdır.

Oracle database'i ile kullanıcı arasında SQL\_NET ilişki kurar. Protokoller belirlenir, bir isim altında konfigürasyonlar birleştirilir. Böylece database ismi tanımlanmasıyla ona ait tüm konfigürasyonlar oluşturulur.

**RDBMS** : Oracle'ın server tarafındaki protokolüdür. Hızlı erişim sağlar. Güvenlik son derece güçlüdür. Client / server desteklidir.

**TCP/IP** : Oracle'ın server tarafındaki protokolüdür. IP numaraları sayesinde client / server mimarisi iletişimi kurulur. RDBMS gibi ortak özellikler taşır.

**Referantional** : Tablolar arasındaki referansların belirtilmesidir. Bazen bir kolonun değerlerinin başka bir tablodaki kolonlardan veri laması gerekmektedir. bu durumda referans verilir.

Bir projeye başlarken şu aşamalar yapılır;

- i.
- ii. İyi bir analiz
- iii.
- iv. Analizin tablolara göre dizaynı ve aralarındaki ilişkiler
- v.
- vi. Akış şeması ve döküman hazırlanması
- vii.
- viii. Test edilmesi
- ix.
- x. Üretime geçilmesi

Database'e üzerinde giriş,değişiklik,silme vb. İşlemlersql ile olur. Tablolar fiziksel olarak gözükmezler. Database içinde saklıdırılar. Operatörler kullanılabilir.

**Data Dictionary**: Database'deki kullanıcılar, yaratılan tablolar vb. Nesneler hakkında detaylı bilgiler bulunan tablodur. Belli başlı fonksiyonlarla ulaşılabilir.

**SQL** : Tablolar arasında iletişim kurar, sorgulama yapar.

**SQL \*Plus** : Ek olarak bloklar oluşturulabiliyor. Mantıksal döngüler ve komutlar oluşturulabiliyor. IF, FOR vb.

**PL / SQL** : Server'a bağlanıp insert, update, delete vb. Database üzerinde direkt işlemler yapılabilir. Database'e en hızlı ulaşım ve sorgulama biçimidir.

- 
- Yazılan komutlar bufferda işlem yapar. O yüzden her yeni komut yazıldığında önceki komutu siler.
- 
- Yazılım kontrolü(syntax) her satırın sonunda kontrol edilir.
- 
- Çeşitli formatlarla çıktı alınabilir.
- 
- Direkt komut sonunda sonuç alınır.,
- 
- Select ile data okunur.
- 
- Insert, update,delete (DML) ile var olan bir tablo üzerinde değişiklikler yapılabilir.
- 
- Create,alter,drop,rename, truncate(DDL) ile yapısal değişiklikler yapılabilir.
- 
- Commit,rollback,savepoint (transaction) ile yapılan işleri onaylar veya geri alır, iptal edilebilir.
- 
- Grant,revoke (DCL) ile objelerin kullanılmasına hak vermek veya geri almak içindir. Güvenlik kurulur.
- 
- Exit (^D) ile SQL'den çıkış sağlanır. Böylece yapılan işlemler iptal olur.

NOT: Tnsnames.ora

☐ Database hakkında bilgiler

Listener.ora

☐ Oracle'da bulunan config dosyasıdır.

NOT: Komut yazılımdaki köşeli parantezler o alanın zorunlu olmadığını belirtir. {} işaretleri o alan için birden fazla değer alacağını gösterir. Büyük harf yazılanlar olduğu gibi yazılır. Küçük harf yazılanlar açıklamadır.

## SELECT komutu

Select komutu ile database'den istenilen kriterlere göre veri getirilir.

Yazılımı:

SELECT [DISTINCT] {\*,column [alias] ,....}

FROM Tablo

[WHERE koşullar]

[ORDER BY {column,expr} [ASC|DESC] ] ;

**SELECT** :Seçilen kolonları alır.

**DISTINCT** :Belirtilen kolondaki aynı değerlere sahip verilerden sadece birini alır. Örneğin personel kodu xxx olanın personel adı gibi.

**alias** :Kolon adlarının daha açık olması için tanımlanır.

**FROM table** :Belirtilen tabloyu açar.

**WHERE** :Belirli kriterler koşullar oluşturulur. Tablolar arasında birleştirmeler yapılabilir.

**ORDER BY** :Seçilen kolona göre sıralı kayıt getirir.

**ASC** :Artan sırada listelenir. Varsayılan budur.

**DESC** : Artan sırada listelenir. Varsayılan budur.

**\*** : Tüm kolonlar seçilmiş anlamındadır.

Listelemede rakamlar sağa yanaşık, karakterler sola dayalı gözükür. Kolon adları büyük harf görüntülenir.Aritmetik işlemler kullanılabilir (+,-,/,\*). Örneğin;

*Select maas+maas\*0.10 From Personel\_Table;*

Bazen bir matematiksel işlemler daha uzun veya daha karmaşık olabilir. Fonksiyon tanımlayarak kolona sorgulatıp bir sonuç üretebiliriz. Matematiksel işlemlerde öncelik sırası;

i.

- ii. Parantezler (,)
- iii.
- iv. \*,/,+,-
- v.
- vi. soldan sağa doğru öncelik sırası,

şeklindedir.

**Column Alias :** Kolon başına açıklayıcı isim verilebilir. Tırnak işaretleri (") ile açıklayıcı kelime belirtilir. Tek kelime için " işareti kullanılması da olur. Örneğin;

*Select ad AS "Adı" From Personel\_Table;*

**Concatenation Operator:** Birden fazla stringi veya kolonu yan yana birleştirip listelemek için kullanılır. İfadeler string olmalıdır. Zaten karakter ve tarih formatındaki değerler tırnak( ' ) işaretleri içerisinde belirtilir. Örneğin;

*Select ad||' '||soyad AS "Adı Soyadı" From Personel\_Table;*

**Null :** Değeri hiç girilmeyen kolonlar NULL değerini alırlar. NULL sıfır veya boşluk karakteri değildir. Yokluk demektir. Bir rakam ile NULL değer üzerinde matematiksel işlemler yapılamaz.

**NVL Fonksiyonu:** Null değeri yerine yeni değer atar. Alan tipi ne türde ise alacağı değer o türde olmalıdır.

Yazılımı:

NVL (null\_değeri\_alan,yeni\_değer)

Örneğin;

*NVL(Maas,0);* □ Maas kolonundaki null değerlerin yerine s 0 atar.

veya `NVL(Ad, ' ');` ☐ Ad kolonundaki null değerlerin yerine bos karakter atar.

En önemli kullanım amacı hesaplamalara işlemin yarıda kesilmesini engellemek içindir. Örneğin maaşa zam miktarı eklensin. Ama bazı personele zam yapılmadığı varsayılırsa bu kişiler için zam kolon değeri null'dur. Buna göre;

`SELECT maas+NVL(zam,0) From Personel_Table;`

**Tab :** Tüm tabloların adlarının saklandığı alandır. Örneğin tüm tabloların listesini vermek için aşağıdaki iki yazılımı da kullanabiliriz.;

Yazılımı;

`SELECT * From Tab;`

`SELECT table_name From user_tables;`

**Desc :** Tablonun kolonlarının adlarını, kolonların tipini ve null değer alıp alamayacağı durumlarını gösterir.

Yazılımı;

`DESC tablo_adı;`

Kolon tiplerinden bazıları;

`NUMBER(rakam,ondalık)`

☐ Sayısal ifadelerdir. Rakam alanı rakamın alabileceği tamsayı kısmını verir. Ondalık alan ise max. decimal değerini verir.

`VARCHAR2(sayı)`  
sayısıdır.

☒ ile belirtilen alan, max. karakter

DATE  
database'de tutulurlar.

☐ Tarih ve saat ifadeler

CHAR(sayı)  
saklanma şeklidir. Max 255 karakter alabilir. Sayı ile belirtilen alan, max. karakter sayıdır.

☐ Karakter ifadesidir. VA

**ORDER BY :** Seçilen kolona göre sıralama yapar. DESC artan sırada, ASC azalan sırada listeler. Null değerler sıralamaya alınmazlar. Örneğin azalan sırada ada göre liste alınsın;

```
SELECT ad,soyad FROM Personel_Table ORDER BY ad DESC;
```

Eğer kolonlar birden fazla ise kolon numarası ile de tanım belirtilebilir;

```
SELECT ad,soyad FROM Personel_Table ORDER BY 1 DESC;
```

Eğer birden fazla kolona göre sıralama yapılmak istenirse, önce yazılan kolon baz alınarak diğer kolona göre sıralı listeler.

```
SELECT ad,soyad FROM Personel_Table ORDER BY 1,2 DESC;
```

**WHERE :** Sorgulamaları sınırlandırmak amacıyla koşullar konulur. Örneğin departmanı 38 kodlu personel listesi;

```
SELECT ad ||' ' ||soyad FROM Personel_Table WHERE Dept_Id=38;
```

<,>, >=, <= gibi karşılaştırma karakterleri kullanılabilir. Bunlar dışında bazı SQL operatörleri ile daha rahat koşullar konulabilmektedir.

**BETWEEN Min AND Max:** Belirtilen Min. ve Max. değerler arasında değer alabilir. Yani belirtilen iki değer arasında değerler alır. Örneğin yaşı 18 ile 30 arasındaki personeller;

```
SELECT ad ||' ' ||soyad FROM Personel_Table
```

*WHERE yas BETWEEN 18 AND 30;*

**IN (liste)** : Liste olarak tanımlanan değerleri alır. Yani bir şekilde OR ifadesine denktir. Örneğin yaşı 18, 25 ile 30 olan personeller;

```
SELECT ad ||' ' ||soyad FROM Personel_Table
```

*WHERE yas=18 OR yas=25 OR yas=30;*

Veya

```
SELECT ad ||' ' ||soyad FROM Personel_Table
```

*WHERE yas IN(18,25,30);*

**LIKE :** Benzerlik vermek için kullanılır. '%' karakteri tüm karakterler yerine geçer. '\_' karakteri ise tek karakter yerine geçer. Sadece \_ işareti çıkararak için '\_' işareti kullanılır. Küçük büyük harf ayrımı vardır.

'%t' □ t harfiyle ba□layanlar.

'%t%' ☐ t harfi geçen tüm kayıtlar.

'B\_K%' ay ve 3 Harfi K an tüm kayıtlar.

karakterleri arasında değer girilirse kolon içinde Örneğin soyadı 'M' ile başlayan personeller;

```
SELECT ad || ' ' ||soyad FROM Personel_Table
```

*WHERE soyad LIKE 'M% ';*



**IS NULL :** Kaydın null olup olmadığını kontrol eder. Örneğin maaşı null olan kayıtları listelesin;

```
SELECT ad || ' ' || soyad FROM Personel_Table
```

```
WHERE maas IS NULL ;
```

**NOT :** Belirtilen koşulun tam ters koşulunu verir. Örneğin maaşı null olmayan kayıtları listelesin;

```
SELECT ad || ' ' || soyad FROM Personel_Table
```

```
WHERE maas IS NOT NULL ;
```

## **SQL \*Plus Üzerinde Sorgu Hazırlanmasında Kullanılacak**

### **Yardımcı Komutlar**

Sql komutları sadece hafızada tutulurlar. İşletilen her komut hafıza tutulur. O yüzden kullanıcının işini kolaylaştırmak için bazı ek komutlara ihtiyaç duyulmuştur.

Sql Plus'a girmek için komut satırında şu şekilde girilir;

```
SQLPLUS [kullanıcı adı [/şifre [@Database]]]
```

**A[PPEND] Text** **S**atır sonuna kelime ekler.

**C[HANGE] /eski/yeni** **E**ski text yerine yeni text ekler. Yeni alanı boş bırakılırsa eski alan olarak belirtilen text silinir.

**CL[EAR] BUFF[ER]** **S**QL hafızasındaki tüm satırları siler, hafızayı boşaltır.

**DEL** **A**ktif olarak bulunduğu satırı siler. Satır no belirtilirse o satırı siler.

**I[INPUT] Text** Satır arasına Text ifadesiyle belirtilen cumleyi ekler.

**L[IST]** Hafızadaki tüm satırları listeler.

**L[IST] m n** Sadece m ile n arasındaki satırları listeler.

**R[UN]** Hafızadaki SQL satırını çalıştırır. Aynı işlevi "R,r,/, ; " karakterleride yapar.

**n Text** n satırındaki cümleyi Text ifadesi belirtilen alana yazılan cümle ile değiştirir.

**O Text** Bulunduğu satırdan bir önceki satıra yeni alan açar ve texti oraya ekler.

**SAVE Dosya** Hafızadaki komutu belirtilen dosya adı altında saklar.

**GET Dosya** Belirtilen dosyayı hafızaya yukler.

**START Dosya** Belirtilen dosyayı direkt çalıştırır. Hafızaya yuklemez. @ işaretide aynıdır.

**ED[IT] Dosya** Belirtilen dosyayı bir editör içinde açar.

**SPOOL** Ekranda yapılan her türlü işi dosyaya atar. SPOOL OFF ile dosyaya kayıt işlemini durdurur.

**EXIT** SQL'den çıkış sağlar.

**HELP Komut** Belirtilen komut için açıklayıcı bilgi verir.

## SQL Plus Fonksiyonları

Fonksiyonlar sayesinde;

- Datalar üzerinde hesap yapabilir,
- Datalar üzerinde değişiklikler yapılabilir,
- Grup oluşturularak bu kayıtlar için kullanılabilir.
- Tarih datası çeşitli formatlarda görüntülenebilir,
- Kolon tipleri değiştirilebilir.
- İç içe fonksiyonlar tanımlanabilir. Öncelikle içteki fonksiyon çalışır. Çıkan sonuca göre dıştaki fonksiyon işlem görür.

**Sys.Dual** : Oracle database'inde bulunan tek kolon ve tek satırdan oluşan bir tablodur. Belirli bir tablodan değerler almadan fonksiyonların kullanılması gerektiğinde bu tablo kullanılır.

### Karakter Fonksiyonları

**LOWER(Text)** : Text olarak belirtilen alanı küçük harfe çevirir.

**UPPER(Text)** : Text olarak belirtilen alanı büyük harfe çevirir.

**INITCAP(Text)** : Text olarak belirtilen alanın baş harfini büyük diğerlerini küçük harfe çevirir.

**CONCAT (Text1,Text2)** : Text1 olarak belirtilen alan ile Text2 alanını birleştirir. ' || ' simgeside aynıdır.

**SUBSTR(Text,m,n)** : Text alanının m. karakterinden itibaren (m. karakter dahil) n kadar karakter alır.

**LENGTH(Text)** : Text alanın karakter uzunluğunu sayısal olarak verir.

**NVL(KOLON,DEĞER)** : Kolonun aldığı değer null ise değer alanında belirtilen değeri verir.

### Sayısal Fonksiyonları

**ROUND(Sayı,m)**: Sayı alanına girilen rakam, m olarak belirtilen ondalık kadar yuvarlar. Örneğin; ROUND(45.923,2)      □ 45.92

ROUND(45.923,0)      □ 46

ROUND(45.923,-1)      □ 50

ROUND(45.951,2)      □ 45.92

ROUND(45.929,2)      □ 45.93

**TRUNC(Sayı,m):** Sayı alanına girilen rakam, m olarak belirtilen ondalık kadar sondan keser.

TRUNC(54.923,2)      □ 45.92

TRUNC(54.923,-1)      □ 40

TRUNC(54.929,2)      □ 45.92

**MOD(m,n) :** m sayısı n sayısına bolundugunde kalan rakamı bulur.

### **Zamansal Fonksiyonları ve İşlemleri**

**Tarih + sayı**    **T**arihe sayı kadar gun ekler.

**Tarih - sayı**    **T**arihe sayı kadar gun çıkarır.

**Tarih - Tarih**    **[k]** tarih arasındaki gün farkını sayısal olarak verir.

**Tarih + sayı / 24**    **T**arihe sayı kadar saat ekler.

**SYSDATE**    **S**istem tarihini verir. Oracle'ın varsayılan tarih formatı 'DD-MON-YY' şeklindedir. Üzerinde aritmatiksel işlemler yapılabilir.

**MONTHS\_BETWEEN(tarih1,tarih2)**    **[k]** tarih arasını ay olarak bulur.

**ADD\_MONTHS (tarih,n)**    **T**arihe n kadar ay ekler.

**NEXT\_DAY(tarih,' gün' )**    **t**arih'den sonraki günün ilk tarihini verir.

**LAST\_DAY(tarih,' gün' )**    **t**arih'den önceki günün ilk tarihini verir.

**ROUND(tarih,[' fmt '])**    **t**arih'i belirtilen formata göre aya veya yıla göre yuvarlar.

**TRUNC(tarih,[' fmt '])**    **t**arih'i belirtilen formata göre aya veya yıla göre keser.

## Çevirme Fonksiyonları

TO\_CHAR(Sayı,['fmt']) : Girilen tarih veya sayıyı karakter tipine çevirir.

Tarih parametreleri;

YYYY ☐ Y ☐ ☐ ☐ rakamsal olarak gosterir.

YEAR  Y  yaz  sal olarak verir.

BC ☐ Millattan önce ve sonrasında için zaman verir.

**MM** **Stein** rakamsal olarak gö

MONTH ☐ Ay ☐ Yıl olarak verir. Belir

☐ ☐ verir.

MON ☐ Ay ☐ yaz ☐ sal sadece ilk 3 harfi

☐ ☐ yaz                  ☒ ☒ ya verir.

RM ☐ Ayırma rakamıyla gösterir.

DD ☐ Günü rakamsal olarak gösterir

DAY  Günü yaz  sal olarak verir.

Sayısal parametreleri;

9. ☐ Rakamd☒r.

1. ☐ S ☐flr ☐rakam ☐verir.

\$  Rakam  ba  ha \$ i  areti koyar.

□ Ondal[k]haneye ay[r].

□ Rakam□'er hane ay□rark aralar□na virgü

MI ☐ İtari koyma ☐ Taraf koyma ☐ İçin sonuna

Örneğin " x nolu kişi xxx'dir." Formatında ekrana liste oluşturalım (id number);

```
SELECT TO_CHAR(id) || ' nolu kişi ' ||name||'dir.'
```

Örneğin sistem tarihini DD-MM-YYYY olarak gösterelim;

```
SELECT TO_CHAR(sysdate,' DD-MM-YYYY')
```

Örneğin sistem zamanını HH:MI olarak gösterelim;

```
SELECT TO_CHAR(sysdate,' HH:MI' )
```

**TO\_NUMBER (Karakter)** : Girilen karakteri sayısal ifadeye çevirir.

**TO\_DATE(char, ['fmt'])** : Girilen karakteri belirtilen formatta tarih formatına çevirir.

Örneğin tarih kolonundaki değerleri, 7 /11/1998 tarihine eşit olanların listesini oluşturalım. Ama kolon değerleri string olarak ayın uzun ad, günü ve yılı sayısal olarak kayıtlıdır. Bu sorunu çözmek için kullanılır;

```
SELECT no, maas
```

```
FROM Personel
```

```
WHERE MAAS_TARIH=TO_DATE('KASIM 7, 1998', 'Month dd,YYYY')
```

### **Tablolar Arasında İlişki Kurulması (JOIN)**

\* Birden fazla tabloyla ilişki kurularak ortak sorgulama yapılmasıdır. Tabloları belli bir ortak kolona göre birleştirmek için join yöntemi kullanılır. Bunun için öncelikle tablolardaki kolonlar aynı tipte ve aynı büyüklükte olmalıdır.

Yazılımı:

```
SELECT table1.kolonları,table2.kolonları
```

```
FROM table1, table2
```

```
WHERE table1.kolon1=table2.kolon2
```

İki tabloyu birleştirmek için iki tabloda ortak bulunan kolonları where kotulu içinde etitleriz.

Örneğin personel adı ve bulunduğu departman listesi için;

```
SELECT a.*,b.*
```

```
FROM personel a , departman b
```

```
WHERE a.dept_id=b.dept_id
```

Böylece personel tablosundan personel adı ve soyadı, departman tablosundan departman adı bilgisi getirilir.

\* Bazı durumlarda tablo birleştirme koşulu bir aralık şeklinde oluşabilir. Yani tabloları birleştirirken anahtar sahaların belirli bir aralığı seçilebilir.

```
SELECT table1.kolonları, table2.kolonları
```

```
FROM table1, table2
```

```
WHERE table1.kolon1 BETWEEN table2.min2 AND table2. max2
```

\* (OUTER JOIN) Bir diğer durumda tablodaki verilerin değerleri null olabilir. Fakat biz null değerlere sahip olanlarıda birleştirmek istersek eksik olan tablonun yanına (+) işareti eklenir.

```
SELECT table1.kolonları, table2.kolonları
```

FROM table1, table2

WHERE table1.kolon1 (+) = table2.kolon2

\* (SELF JOIN) Bir başka durumda aynı tablodaki kendi içindeki kolonlar arasında ilişki kurulabilir. Bazı durumlarda kolonlardaki değerleri eşit olma durumlarına göre sorgu düzenlenebilir.

SELECT table1.kolonları

FROM table1 a, table1 b

WHERE table1.a = table1. b

### **Grup (GROUP) Fonksiyonları**

Bazen belli bir koşula uygun belli bir topluluk için sorgu kurulabilir. Bu durumda grup işlemi yapılır. Örneğin departmanlara göre grup oluşturulup o departmandaki personel sayısı hesaplanabilir. Grup fonksiyonları sadece grup ifadeleriyle kullanılabilir.

#### Yazılımı:

SELECT kolonlar, grup fonksiyonları

FROM table

[WHERE kotul]

[GROUP BY grup\_kolonu]

[HAVING group\_kotulu]

[ORDER BY kolon]

GROUP BY satırı ile kolonlar üzerinde küçük gruplar oluşturulur.



HAVING satırı ile bu oluşturulan grup için grup koşulları tanımlanabilir. Where koşulundan farkı WHERE kayıt üzerinde koşul koyar, HAVING ise sadece grup kayıtları üzerinde koşul konabilir ve grup fonksiyonları kullanılabilir.

**AVG(Kolon)** : Belirtilen kolonun ortalamasını bulur.

**COUNT(Kolon)** : Belirtilen kolonun kayıt sayısını bulur. Count(\*) ile o grupta oluşturulan kayıt sayısı hesaplanır. Null değerler için nvl fonksiyonu kullanılır. Çünkü kolon içindeki null değerler işleme alınmaz.

**MAX(Kolon)** : Belirtilen kolondaki kayıtların değerlerinin maksimum değerini bulur. Sayısal, karakteristik veya tarihsel olarak kendi içinde sıralama yapabilir.

**MIN(Kolon)** : Belirtilen kolondaki kayıtların değerlerinin minimum değerini bulur. Sayısal, karakteristik veya tarihsel olarak kendi içinde sıralama yapabilir.

**SUM(Kolon)** : Belirtilen kolondaki kayıtların değerlerinin sayısal toplamını bulur.

**VARIANCE(Kolon)** : Belirtilen kolondaki kayıtların değerlerinin matematiksel varyansını bulur.

Örneğin bir personel listesindeki departmanların ayrı ayrı departman içindeki personelin max ve min maas alan kişilerin maaslarını isteyelim;

```
SELECT Departman_id, MAX(Maas) , MIN(maas)
```

```
TABLE Personel
```

```
GROUP BY Departman_id
```

Veya 38 nolu departmandaki personel sayısını bulalım;

```
SELECT COUNT(*)
```

```
TABLE Personel
```

```
WHERE Departman_id = 38
```

Veya tüm departmanlardaki maaşlarının ortalaması 80.000.000 'den büyük olanların listesini oluşturalım;

```
SELECT Departman_id, AVG(Maas)
```

```
TABLE Personel
```

```
GROUP BY Departman_id
```

```
HAVING AVG(maas) > 80000000
```

Bazı durumlarda iç içe grup oluşturulması istenebilir. Bu durumda şu şekilde yazılır.

```
GROUP BY Kolon1, Kolon2 , ...
```

Örneğin Departman adına ve personel yasına göre grup oluşturalım.

```
SELECT Departman_id, Yas
```

```
TABLE Personel
```

```
GROUP BY Departman_id, Yas
```

### **Altsorgular(SUBQUERY)**

Subquery, bir sorgu oluşturulurken, bu sorguya ait kriterleri başka bir sorgu belirleme durumudur. Kısaca sonucu bilinmeyen koşullar olduğunda bu yöntem kullanılır.

Yazılımı:

```
SELECT .....
```

```
FROM .....
```

```
WHERE kolon=( SELECT ...
```

FROM ....

WHERE ....)

Select cümlesi parantes içinde tanımlanır. = koşulu konu ise select cümlesi tek değer döndürmek zorundadır. Eğer birden fazla değer için koşul konulması gerekirse " in, not in " kelimesi kullanılır. Tek kayıt döndüğünde = yerine >, >=, <, <=, <> kontrol ifadeleride kullanabiliriz. Bir önemli nokta da koşula eşitlenecek değer 2. Select cümlesi içindeki kolon değerine eş olmalıdır. Yani tipleri eşit olmalıdır.

Subquery sorgulama tekniğinde order by kullanılamaz.

Having grup kotulu içinde select cümlesi kullanabiliriz.

Örneğin;

1. sorgu : Kazım' ın departman numarası kaçtır?
2. SELECT dept\_id,name

FROM Personel

WHERE name='Kazım'

3. sorgu : Departman numarası 38 olan kişilerin listesinden adı kazım olan var mı incele?
4. SELECT dept\_id,name

FROM Personel

WHERE dept\_id=38

5. sorgu : Bu iki sorgunun birleşimi istenirse, yani departmanı 38 olan kazım adlı personel bigisi ?

☐ ☐ SELECT dept\_id,name

FROM Personel

WHERE dept\_id=( SELECT dept\_id

□ □ □ □ FROM Personel

WHERE name='Kazım')

### **Çalışma Anında Sorguya Dışarıdan Değer Alma**

Bir sorgu oluşturulurken bazen bazı alanların değerleri değişkendir. Bu yüzden sorguya dışarıdan değer girilmesi ve bu değere göre bir sorgu oluşturulması gerekir. Bunun için anahtar kelimenin başına ' & ' karakteri getirilir.

Örneğin istenilen departmankoduna göre personelin adı ve soyadı listeleyelim;

SELECT ad,soyad

FROM Personel

WHERE Dept\_id = &Departman\_kodu

Ekran:

Departman\_kodu için değer gir: 38

eski 1: select ad,soyad from Personel where Dept\_id = &Departman\_kodu

yeni 1: select ad,soyad from Personel where Dept\_id = 38

Eğer sorgu yazılımı sorunlu ise bu giriş yapıldıktan sonra anlaşılır ve kesinlikle girilen değer kolon tipine eş olmalıdır. Örneğin '&anahtar\_kelime' şeklinde tırnak arasında belirtilmişse giriş olarak sadece text değer girilir. Ama &anahtar\_kelime tırnak içinde belirtilmemişse giriş yapılırken tırnak içinde giriş yapılmalıdır. Birden fazla &anahtar\_kelime tanımlanırsa sırayla sorgu sorulur ve tüm değerler girildikten sonra sorgu oluşturulur. Eğer where koşulunda &anahtar\_kelime'si kullanılmışsa giriş yapılırken matematiksel koşul şeklinde giriş yapabiliriz.

Eğer anahtar kelimenin başına ' && ' konulursa sorgu sırasında sadece bir kerelik sorgu sorulur. Bundan sonra anahtar kelimenin değeri ilk girilen değer olarak geçerlidir.

**DEFINE değişken=değer** : Belirtilen değişkene karakter atanır.

**ACCEPT** : Kullanıcıya açıklayıcı mesaj verilerek istenilen tipte veri girişi yapılması sağlanır. & işaretiyle yapılan işlemin daha kontrollü şeklidir.

Yazılımı :

ACCEPT değişken [datatipi] [ FORMAT ] [PROMPT text] [HIDE]

Değişken

☐ Giriş yapacak verinin

Datatipi

☐ Giriş yapacak verinin veri tipidir.

FORMAT

☐ rakam için 9999 verinin giriş düzeyi

(4 haneli sayı) , karakter için A10 (10 karakterlik yer) .

PROMPT

☐ Kullanıcıya gireceği değeri

HIDE

☐ Giriş yaparken verinin

Örneğin şifre girişlerinde.

Örnekler;

ACCEPT adi PROMPT 'Adınızı giriniz:'

ACCEPT yas NUMBER PROMPT 'Yaşınızı giriniz:'

ACCEPT yıl NUMBER FORMAT 9999 PROMPT 'Bulunduğunuz yılı giriniz:'

ACCEPT sifre CHAR PROMPT 'Tifrenizi giriniz:' HIDE

Örnek: Personel adına göre personel bilgilerini gösterelim. Bunun için bir .sql uzantılı dosyada yazalım. Bu programı start ile çalıştıralım.

Ornek.sql

SET ECHO OFF

ACCEPT adi PROMPT ' Adınızı Giriniz:'

```
SELECT * FROM Personel
```

```
WHERE UPPER(Personel.ad) LIKE UPPER('&adi')
```

```
/
```

```
SET ECHO ON
```

“Start Ornek.sql “ yazılarak sql çalıştırılır.

Bazen parametre girişlerini daha başlangıçta vermek istenirse;

Start Dosya\_adı Parametreler,...

Örneğin yukarıdaki örnek için Start Ornek Kazım

### **Tabloların Yaratılması**

Database’de verilerin saklanması amacıyla tablolar yaratılır. Tablo yaratabilmek için o kullanıcının buna yetkisi olmalıdır. Aynı zamanda limitsiz tablespace hakkına sahip olmalıdır.

Yazılımı :

```
CREATE TABLE [kullanıcı.]table
```

```
(kolon datatipi(boyutu) [DEFAULT değer]
```

```
[CONSTRAINT constraint_name] constraint_type]
```

```
[table_constraint]
```

```
[PCTFREE integer] [PCTUSED integer]
```

```
[TABLESPACE tablespace] [STORAGE storage_clause]
```

```
[CACHE]
```

[ENABLE enable\_clause]

[DISABLE disable\_clause]

[AS subquery]

**Kullanıcı** Tablonun yaratıcısıdır. Onun sahip olduğu haklara ve mekana sahiptir.

**Kolon** Tablonun kolon adlarını belirler. Bir tablodaki kolon sayısı 1-254 arasında olmalıdır.

**Datatipi** Bir kolonun veri tipini belirler.

☐ ☐ Varchar2(boyut): Boyut ile belirtilen max miktar kadar karakterdir.Max  
☐ ☐ de ☐ ri 2000'dir.

Char(boyut) : Boyut ile belirtilen max miktar kadar karakterdir.Max.  
☐ ☐ de ☐ ri 255'dir.

Number :  $e^{38}$  'e kadar olan tüm sayısal değerlerdir.

☐ ☐ Number(m,n) : m kadar ( $\max e^{38}$ ) sayının n kadar ondalık alan için  
değer alır.

Date : Tarih ve saat değerlerini bir tutar.

Boolean : Mantıksal ifadeleri saklar. Yani doğru ise True,

☐ ☐ ☐ ☐ yanl ☐ se false.


Long : Max 2GB 'a kadar büyük olan alanlar için yer tutar.


Raw : Grafikselsel yapıdaki veriler için tanımlanır.


**DEFAULT** Herhangi bir kolona değer girilmezse, direkt olarak değer atamaya yarar. Girilen değer veri tipi ile aynı olması gerekir.


**CONSTRAINT** Bir kolon için daha açıklayıcı olması açısından ve yapısı açısından kullanılır.


**table\_constraint**  tablo için daha açıklayıcı olması açısından belirtilen etikettir.

**PCTFREE**  Tablo üzerinde yapılan kayıt güncelleme işlemleri için, kayıt bloklarındaki ayrılacak rezervasyon yüzde değeridir. 1-99 arasında bir tamsayıdır. 0, tüm bloklara yeni kayıt girişine izin verilir. Varsayılan değer 10'dur. Yani güncelleme için her bloğun %10'unu rezerve eder. Geri kalan %90'nını ise yeni kayıt girişlerine ayırır.


**PCTUSED**  Tablonun her veri bloğu için Oracle'nin koruduğu, kullanılmış alanın min yüzdesini verir. 1-99 arasında bir tamsayıdır. Varsayılan değeri 40'dır. Verilen parametre değerinin altına düşek blok için, yeni kayıt girişi seçimlidir. PCTFREE+PCTUSED <100 olmak zorundadır.


**TABLESPACE**  Oracle'ın tabloyu yaratacağı tablo boşluğunu belirler. Eğer tanımlanmazsa çalışılmakta olan kullanıcının tablo boşluğunda yaratılır.

**STORAGE**  Tablonun depolama karakteristiğini belirler. Büyük tablolarda performans düzenleyici olarak kullanılır.

**CACHE**  Tablo üzerinde arama yapılırken, hafıza cache'nin en çok kullanılanlar üzerinde yer almasını sağlar.

**ENABLE**  Bütün sınırlamaları geçerli kılar.

**DISABLE**  Bütün sınırlamaları iptal eder.

**AS query**  Tablo yaratılırken, alt sorgu aracılığı ile geriye döndürülen satırlar tabloya yerletirilir.(insert)

## Data Dictionary

Database ile ilgili her türlü bilgiyi buraya atar. Buradan verilere ulaşabiliriz. Kullanıcılar, tablolar, indexler, constraintsler vb. birçok açıklayıcı bilgiler tutulur. Bütün tablo ve viewerların adlarını ve açıklamalarını örnek için ;

```
SELECT * FROM Dictionary;
```

Dictionary'nin yapısını görmek için;

```
DESC dictionary
```

İsim Tip



-----  
TABLE\_NAME VARCHAR2(30)

COMMENTS VARCHAR2(2000)

Dictionary'de bulunan objelerden bazıları;

**User\_tables**    Hangi kullanıcı ile login olundu ise o kullanıcıya ait tabloları listeler.

**All\_tables**    Bütün kullanıcıların yarattığı tabloları listeler.

**Db\_tables**    Database ile ilgili her türlü tablolar listelenir.

**V\$\_tables**    Sistemin performansını ölçmek için tutulan bilgilerdir.

**V\$\_librarycache**    Cache'in yeterli olup olmadığını anlamak için bakılır.

**V\$\_rollstat**    Rollback segmentinin yeterli olup olmadığını anlamak için bakılır.

Örnek : Login şifresiyle giren kullanıcının yarattığı tabloların listesini alalım;

```
SELECT table_name FROM user_tables
```

Örnek: Tüm obje isimlerinin listesini alalım;

```
SELECT object_name FROM user_objects
```

## **Database Üzerinde Transaction Kontrol Komutları**

Transaction, datanın tutarlı şekilde korunmasını sağlayan yöntemdir. Database üzerinde kayıt yapma , silme, düzeltme vb. işlemlerin yapılabilmesi için aşağıdaki komutlara ihtiyaç vardır;

**INSERT INTO** : Tabloya yeni bir kayıt eklenmesini sağlar.

Yazılımı;

INSERT INTO table [(kolon1, kolon2, ...)] VALUES (değer1, değer2, ...)

**Kolon** Tablonun kayıt yapılacağı kolon adlarını belirler. Bir tablodaki kolon sayısı 1-254 arasında ve o tabloda var olmalıdır.

**Değer** Kolon için alacağı değer verilir. Kolon tipiyle aynı olmalıdır. Eğer kolon adları belirtilmezse değerler kolon sırasına göre değer almalıdır. Değerlere sysdate, user vb. hazır fonksiyonlar aktarılabilir. Eğer bir alana değer verilmesi istenmiyorsa, yani boş değer verilmesi isteniyorsa NULL değeri verilir.

Örnek: Personel tablosuna yeni bir personel girişi yapalım;

```
INSERT INTO personel (ad,soyad,maas,ayrilma_tarihi,kayit_giris_tarihi)
```

```
VALUES ('Kazım','Sarıkaya',400000000,NULL,SYSDATE)
```

Aynı örnek için kolon sırasını bildiğimiz düşünülürse şu şekilde yazılabilir;

```
INSERT INTO personel
```

```
VALUES ('Kazım','Sarıkaya',400000000,NULL,SYSDATE)
```

Sadece tek kayıt girişi yapılıyorsa, rahat veri girişi yapılması için değerlere değişken tanımlayarak veri girişi sağlayabiliriz. Böylece table her çalıştığında yeni veri girişi yapılır.

Örnek: Personel tablosuna her çalıştırdığımızda yeni bir personel girişi yapalım;

```
INSERT INTO personel (ad,soyad,maas,ayrilma_tarihi,kayit_giris_tarihi)
```

```
VALUES ('&Adi','&Soyadi',&maas,NULL,SYSDATE)
```

Örnek: Daha güvenli ve rahat giriş yapılacak şekilde kayıt yapılacak script yazalım;

```
SET ECHO OFF
```

ACCEPT ad PROMPT 'Adını giriniz:'

ACCEPT soyad PROMPT 'Soyadı giriniz:'

ACCEPT maas PROMPT 'Maaşını giriniz:'

INSERT INTO personel(ad,soyad,maas,ayrilma\_tarihi,kayit\_giris\_tarihi)

VALUES ('&ad','&soyad',&maas,NULL,SYSDATE)

/

SET ECHO OFF

Kayıt yapılacak kolonların değerleri bilinmiyorsa veya başka bir tablodan değer alınması gerekiyorsa aşağıdaki yapı kullanılır;

Yazılımı;

INSERT INTO table [(kolon1, kolon2, ...)] Altsorgu

Örnek: Personel tablosuna departman tablosundan departman kodu 38 olan personellerin değerlerini aktaralım;

INSERT INTO personel(ad,soyad,maas,ayrilma\_tarihi,kayit\_giris\_tarihi)

SELECT adi,soyadi,maasi,ayrilma,kayit\_giris

□ □ FROM departman

WHERE dept\_id=38

**UPDATE :** Tablodaki kolonların değerlerini değiştirmek için kullanılır.

Yazılımı;

UPDATE table

SET kolon1=değer1, kolon2=değer2, ...

[WHERE kotul]

**Kolon** Tabloda değişiklik yapılacak kolon adıdır. Bir tablodaki kolon sayısı 1-254 arasında ve o tabloda var olmalıdır.

**Değer** Kolon için alacağı değer verilir. Kolon tipiyle aynı olmalıdır. Eğer kolon adları belirtilmezse değerler kolon sırasına göre değer almalıdır. Değerlere sysdate, user vb. hazır fonksiyonlar aktarılabilir. Eğer bir alana değer verilmesi istenmiyorsa, yani boş değer verilmesi isteniyorsa NULL değeri verilir.

Örnek: 41 nolu departmanın maaşını %13 artıralım;

```
UPDATE Personel
```

```
SET maas=maas+maas*13/100
```

```
WHERE dept_id=41
```

Örnek: 649 nolu personelin departman kodu 10 olsun;

```
UPDATE Personel
```

```
SET dept_id=10
```

```
WHERE personel_id=649
```

Eğer WHERE koşulu konulmazsa tüm tablo için kolon değerini değiştirir. Koşulu SET ile değiştirdiğimiz kolona veremeyiz.

**DELETE** : Tabloda belirtilen kayıdı siler.

Yazılımı;

```
DELETE FROM table
```

```
[WHERE kotul]
```

Örnek : Personelin maaşları 100.000.000 'dan küçük olan kayıtları silelim;

```
DELETE FROM Personel
```

WHERE maas<100000000

Bir tablodaki tüm kayıtları silmek istersek koşul konmaz. Tablo silinmez, sadece kayıtları silinir.

Örneğin DELETE FROM Personel

Foreign key veya primary key silinecek kayıtda varsa silinme hatası verir. Çünkü başka tablo ile ilişki kurulduğundan(join), diğer tablonun kullandığı kayıt vardır. O yüzden bu kaydı silebilmek için, ilişkide bulunduğu tüm kayıtları önce silmek gerekir. Aynı zamanda tablo üzerinde çalışan kullanıcı varsa kayıtlar silinemez. ROLLBACK komutu ile silinen kayıtları kurtarabiliriz(Tabi ki en son COMMIT komutu kullanılana kadar).

**COMMIT** : Bütün yapılan işlemleri kesin olarak kalıcı olmasını sağlar. Böylece yapılan değişiklikleri varsa diğer kullanıcılarda görür.

**SAVEPOINT x** : İşlemi belirli bir yere yönlendirmek için kullanılır. X ile belirtilen alan için işaret konularak istenildiğinde bu işarete kadar işlemler yapılabilir.

**ROLLBACK x** : Bütün yapılan işlemleri kesin olarak iptal eder. SAVEPOINT komutu ile belirlenen x kodlu alana kadar olan tüm itleri iptal eder.

Yazılımı;

ROLLBACK [TO SAVEPOINT x]

Örnek:

UPDATE personel .....

SAVEPOINT dön

DELETE FROM Personel

INSERT INTO .....

ROLLBACK dön teklindeki bir işlemle personel kaydının silinmesi ve yeni kayıt eklenmesi işlemi iptal edildi. Ama ilk yapılan değişiklik kaldı.

**Read uncommitted** Commit olmadan görünmez. O halde böyle level yoktur.

**Read committed** Oracle'ın varsayılan değeridir. Committedten sonra işlem görür.

**Serializable** Yapılan insert işlemleri gözükmez. Eski datalar gözükür.

**Read only** Transaction itleminde sadece okuma var demektir.

**ALTER TABLE :** Yeni bir kolon eklemek, kolonun tipini veya uzunluğunu değiştirmek vb. yapısal değişiklikler yapılması için kullanılır. Eğer kolon üzerinde değişiklikler yapılacaksa dikkat edilmesi gereken koşullar vardır. Örneğin kayıt uzunluğu 15 iken uzunluğunu 10'a indirsek kayıt içindeki bilgiler kesilir. Kolonlar ekleyebilir ve yapısal değişiklikler yapabiliriz. Constraint yapısını ekler, silebilir, enable ve disable yapabiliriz.

Yazılımı;

ALTER TABLE table

[ADD (kolon datatipi [DEFAULT değer] [NOT NULL]) , ...]

[ADD [CONSTRAINT açıklama] tipi (kolon)]

[MODIFY (kolon datatipi [DEFAULT değer] [NOT NULL]) , ...]

[DROP [CONSTRAINT açıklama] tipi (kolon)]

[ENABLE | DISABLE CONSTRAINT açıklama]

**ADD** Yeni bir kolon ekler.

**MODIFY** Kolonun içeriğini, uzunluğunu ve tipini değiştirir. Bunun için kayıtların değeri null olmalıdır.

**DROP** Kolonu siler.

**Kolon** Tabloda işlem yapılacak kolon adıdır.

**Datatipi** Kolonun alacağı tipi belirler.

**DEFAULT değer** Kolonun ilk alacağı değerdir. Kolon hiç değer atanmaz ise burada değer ile ifade edilen sabit veri aktarılır.

**ENABLE** Unique veya primary key indexlerin otomatik yaratılmasını sağlar. Bütün kayıtların tanımlanan unique veya primary key tanımına uyması gerekir.

**DISABLE** Unique veya primary key tanımını kaldırır.

Örnek : Personel tablosuna ek maas isminde 15 uzunluğunda bir kolon yaratalım. İlk değer 0 olsun.

```
ALTER TABLE Personel
```

```
ADD (ekmaas number(15) DEFAULT 15)
```

Örnek : Personel tablosundaki pers\_id ile departman tablosundaki dept\_id arasında FOREIGN KEY kuralım.

```
ALTER TABLE Personel
```

```
ADD CONSTRAINT personel_dept_id_fk
```

```
FOREIGN KEY (dept_id)
```

```
REFERENCES personel(pers_id)
```

Örnek : Personel tablosundaki maas kolonundaki alan uzunluğunu 15'e çıkaralım.

```
ALTER TABLE Personel
```

```
MODIFY (maas number(15))
```

Örnek : Personel tablosundaki maas kolonu silinsin.

```
ALTER TABLE Personel
```

```
DROP (maas number(15))
```

Örnek: Personel tablosundaki personel\_id primary key için index yaratılması otomatikleştirilim;

```
ALTER TABLE Personel
```

```
ENABLE CONSTRAINT Personel_id_pk
```

**DROP TABLE :** Tabloyu fiziksel olarak siler. Rollback komutu ile silinen tablo geri getirilemez. Tabloyu ancak yetkisi olan kullanıcı silebilir.

Yazılımı;

```
DROP TABLE table
```

```
[CASCADE CONSTRAINTS]
```

Örnek: Personel tablosunu silelim;

```
DROP TABLE Personel
```

**RENAME ..TO.. :** Objelerin ismini değiştirmek için kullanılır. Otomatik olarak commit olur. Tabloyu ancak yetkisi olan kullanıcı silebilir.

Yazılımı;

```
RENAME eski_isim TO yeni_isim
```

Örnek: Personel tablosunun adını pers olarak değiştirelim;

```
RENAME personel TO pers
```

**TRUNCATE TABLE :** Tablodaki tüm kayıtları siler. Delete komutu gibi olmasına karşın o komuttan çok daha hızlı silme işlemi yapar. Rollback komutu ile silinen kayıtlar geri getirilemez. Otomatik olarak commit olur. Tabloyu ancak yetkisi olan kullanıcı silebilir.



Yazılımı;

TRUNCATE TABLE table

Delete komutu ile Truncate arasındaki en önemli fark; Delete komutu kayıtları silmek için kayıtlarda boşluk bırakır. Truncate ise tamamen kayıtları temizler, yani başa sarar.

Örnek: Personel tablosundaki tüm kayıtları silelim;

TRUNCATE TABLE Personel

**COMMENT ON TABLE :** Tablo veya kolonların içeriği hakkında açıklayıcı bilgi verilir. 2000 byte açıklama alanı girilebilir.

Yazılımı;

COMMENT ON TABLE table | COLUMN table.kolon

IS 'açıklama'

Örnek : Personel tablosu açıklaması 'Başak Sigorta'da çalışan kişiler' olarak not düşelim;

COMMENT ON TABLE personel

IS 'Başak Sigorta'da çalışan kişiler'

Örnek : Personel tablosundaki pers\_id kolonun açıklaması 'Personel numarası' olarak not düşelim;

COMMENT ON COLUMN personel.pers\_id

IS 'Personel numarası'

**ALL\_COL\_COMMENTS** Yetkili olan kullanıcının kolonlarının açıklayıcı notları listelenir.

**USER\_COL\_COMMENTS** Tüm kullanıcıların kolonlarının açıklayıcı notları listelenir.

**ALL\_TAB\_COMMENTS** Tüm kullanıcıların tablolarının açıklayıcı notları listelenir.

**USER\_TAB\_COMMENTS** Yetkili olan kullanıcının tabloları için açıklayıcı notları listelenir.

### Otomatik Numara Üretimi(Sequence)

Yaptığı iş unique sayılar üretmektir. Belli oranlarda arttırmalar yapılmakta kullanılır. Her çağrıldığında yeni bir sayı üretir. Extra bir hesaplama yapılmadan, seri olarak tanımlandığı şekilde rakamlar üretir. Sayıları cacheden okuduğu için çok hızlı sonuç üretir.

#### Yazılımı;

CREATE SEQUENCE kolon

[INCREMENT BY n]

[START WITH n]

[MAXVALUE n | NOMAXVALUE]

[MINVALUE n | NOMINVALUE]

[CYCLE | NOCYCLE]

[CACHE n | NOCACHE]

**INCREMENT BY n** Sayının artış miktarı belirtilir. Varsayılan değer 1'dir. Birer birer sayı artırılır.

**START WITH n** Numaranın üretileceği başlangıç numarasıdır. Varsayılan değer 1'dir.

**MAXVALUE n** Numaranın alabileceği maksimum rakam yazılır.  
NOMAXVALUE ile bitiş numarası verilmez. Max =  $10^{27}$

**MINVALUE n** Numaranın alabileceği minimum rakam yazılır. NOMINVALUE ile başlangıç numarası verilmez. Min = 1

**CYCLE | NOCYCLE** Maxvalue değeri kadar işlem yapıldıktan sonra işlemi tekrardan başlatır. Varsayılan değer NOCYCLE'dir.

**CACHE** Cache sayısı kadar sayıyı hafızaya gönderir. Varsayılan değer NOCACHE=20'dir.

Örnek: Personel departmanının pers\_id kolonunu 10'dan 50'ye kadar 2'şer 2'şer arttıralım;

```
CREATE SEQUENCE pers_id
```

```
INCREMENT BY 2
```

```
START WITH 10
```

```
MAXVALUE 50
```

```
NOCYCLE
```

```
NOCACHE
```

Örnek: Personel departmanının pers\_id kolonunu birer birer arttıralım;

```
CREATE SEQUENCE pers_id
```

```
INCREMENT BY 1
```

```
START WITH 1
```

```
NOMAXVALUE
```

```
NOCYCLE
```

```
NOCACHE
```

**USER\_SEQUENCES** : Sequences'in en son durumu gösterir.

Yazılımı;

```
SELECT sequence_name, min_value, max_value, increment by, last_number  
FROM user_sequences
```

**Sequence\_name** İşlemlerin tanımlandığı isimdir.

**Min\_value** Sayının aldığı min değerdir.

**Max\_value** Sayının aldığı max değerdir.

**Increment\_by** Sayının aldığı artış miktarının gösterir.

**Cycle\_flag** İşlemin tekrar yapılması isteniyorsa Y, istenmiyorsa N değerini alır.

**Cache\_size** Hafızadaki cache sayısıdır.

**Last\_number** Numaranın alacağı değerdir

**NEXTVAL** : En son aldığı sayının bir sonra alacağı değeri gösterir.

Örnek: Personel tablosuna pers\_id kolonuna otomatik değerler tatarak kayıt edelim;

```
INSERT INTO personel
```

```
VALUES (pers_id.NEXTVAL,'Kazım','Sarıkaya')
```

**CURRVAL** : O anki sayısal değeri gösterir. Eğer hiç nextval fonksiyonu kullanılmamışsa değeri boş değer gelir.

Örnek: pers\_id kolonunun o anki alacağı değeri seçelim;

```
SELECT pers_id.CURRVAL  
FROM DUAL
```

**ALTER\_SEQUENCE** : Değişiklik bir yerden başlanacaksa önce drop edilir, sequence işlemini değiştirir.

Yazılımı;

```
ALTER SEQUENCE kolon  
  
[INCREMENT BY n]  
  
[MAXVALUE n | NOMAXVALUE]  
  
[MINVALUE n | NOMINVALUE]  
  
[CYCLE | NOCYCLE]  
  
[CACHE n | NOCACHE]
```

**DROP\_SEQUENCE** : Kolon için yaratılan işlemleri siler.

Yazılımı;

```
DROP SEQUENCE kolon
```

### **View**

Bir tablo üzerinde sorgulama yapılması için kullanılan nesnedir. Fiziksel olarak herhangi bir yerde saklanmaz. Avantajları;

- Database erişimini kısıtlar. Böylece sadece sorgulanan verileri gözükür.
- Sorgulamaları kolaylaştırabilir.
- Datayı bağımsız olarak gösterebiliriz.

Yazılımı;

```
CREATE [OR REPLACE] [FORCE | NOFORCE]
```

VIEW view\_adi [alias]

AS subquery

[WITH CHECK OPTION [CONSTRAINT constraint]]

[WITH READ ONLY]

**FORCE** Hata durumu oluşursa gözardı ettirilir. NOFORCE ise gözardı edilmez.

**View\_adi** Yaratılan view sorgu adıdır..

**Alias** Yaratılan işlemin adıdır.

**WITH CHECK OPTION [CONSTRAINT** View objesine hatalı işleme yapılmasını engeller.

Örnek: Departman numarası 41 olan peroneller için perview isminde bir view olşturalım;

```
CREATE VIEW persview
```

```
AS SELECT *
```

```
FROM personel
```

```
WHERE dept_id=41
```

```
WITH CHECK OPTION CONSTRAINT empview_ck;
```

Departman id'si 16 olan personellerin departman numarasını 38 yapalım;

```
UPDATE persview
```

```
SET dept_id=38
```

```
WHERE id=16;
```

Bu durumda bu değişiklik tabiki yapılamaz.

**WITH READ ONLY** View objesinin sadece okuma amaçlı olduğu belirtilir. Farklı bir işlem yapıldığında bunu engeller. Aslında kayıtlar üzerinde değişiklik yapılmasını engeller.

Örnek: Departman numarası 45 olan peroneller için persview isminde bir view oluşturalım;

```
☐ ☐ CREATE VIEW persview
```

```
AS SELECT *
```

```
FROM personel
```

```
WHERE dept_id=41
```

```
WITH READ ONLY;
```

Departman id'si 10 olan personellerin silinmesini sağlayalım;

```
DELETE FROM persview
```

```
WHERE id=10;
```

Bu durumda işlemi keser ve uyarı verir.

**USER\_VIEWS** : Data dictionary tablosunda view tanımları bulunur. View adı, text uzunluğu ve text'i saklar.

VIEW\_NAME

☐ Kullanıcıya ait view adları n karakter tutar.

TEXT\_LENGTH

☐ Sql ifadenin uzunluğudur.

TEXT

☐ Sql ifadesidir.

**DROP VIEW** : Yaratılan view objesini siler.

Yazılımı;

```
DROP VIEW view_adı
```

## INDEX YARATMA

Indexler, bir tablonun istenilen kolonlarına daha hızlı erişim olanağı sağlamak için kullanılır. Tablodaki kayıtlar üzerinde giriş/çıkış işlemleri yapılırken dataya daha hızlı ulaşılması sağlanır. Primary key tanımlanan kolonlar için otomatik olarak index yaratılır. Index en fazla 16 kolondan oluşur. Bir kolon tipi long veya long raw olamaz.

Yazılımı;

```
CREATE INDEX index_adı
```

```
ON tablo_adı(kolonlar)
```

Örnek: Personelin adına ve soyadına göre index oluşturalım;

```
CREATE INDEX personel_inx
```

```
ON personel(ad,soyad);
```

Böylece ad kolonu öncelikli olmak kaydıyla birlikte soyadına göre sıralama yapar, yani index oluşturur. Önce ada göre sıralar, eğer aynı isimden birden fazla kayıt olursa bu sefer soyad kolonundaki değerlere göre sıralama oluşturur.

Index yaratabilmek için;

- Index yaratılacak tablonun var olması gereklidir,
- Index yaratma hakkına sahip olmalıdır,
- Limitsiz tablespace hakkına sahip olmalıdır.



Index yaratılma ihtiyacı aşağıdaki koşullar oluştuğunda belirir;

- Where ifadesi ile sık sık kullanılan kolonlar kullanılıyorsa,
- Join ile birleştirme işlemi yapıldığında,
- Kolon değerleri geniş aralıkta ise,
- Büyük rakamlı kolonların içinde null değerler bulunuyorsa.

Index yaratılmama ihtiyacı aşağıdaki koşullar oluştuğunda belirir;

- Tablo küçük ise,
- Tablodaki bazı kolonlara sık sık ihtiyaç duyulmadığında,
- Tabloda sık sık değişiklikler yapılıyorsa.

**USER\_INDEXES** Index isimlerini ve unique olup olmadığı bilgilerini saklar.

Örnek: Personel tablosuna ait index dosyalarını listeleyelim;

```
SELECT ic.index_name,ic.column_name,ic.colum_position,ix.uniqueness
```

```
FROM user_indexes ix, user_ind_columns ic
```

```
WHERE ic.index_name=ix.index_name
```

```
AND table_name='Personel';
```

**DROP INDEX** Yaratılan indexleri veri tabanından yok etmek için kullanılır.

## DATABASE GÜVENLİĞİ

Database güvenliği için kullanıcılar tanımlanır. Her kullanıcının bir şifresi bulunur. Kullanıcılara belirli yetkiler verilir.

Sistem ve data güvenliği olmak üzere iki kısımda incelenir. Sistem güvenliği içinde kullanıcıya yaratma, değiştirme, silme vb. yetkiler tanımlanır. Data güvenliği için ise datalar başka bir ortama yedekleme(backup) yapılarak sağlanır.

Database level aşamaları ile güvenlik sınırlanır. Eğer yüksek level yetkisi varsa yeni kullanıcı(veya tablolar) yaratabilir veya silinebilir. Tabloların yedekleri alınabilir.

**CREATE USER** Database üzerinde işlemler yapılabilmesi için kullanıcı yaratılır.

Yazılımı;

CREATE USER user\_adı

IDENTIFIED BY tıfre

**DROP USER** Databaseden kullanıcıyı siler.

Yazılımı;

DROP USER user\_adı

**GRANT (Yetki)** Kullanıcıya yetki verilmesi için kullanılır.

Yazılımı;

GRANT yetkiler\_roller

TO kullanıcı [PUBLIC]

[WITH GRANT OPTION]

[PUBLIC]

WITH GRANT OPTION

vermeyi sağlar. Bir rolü bu şekilde belirlersek kullanıcı rolleri değiştirebilir veya silebilir.

☐ Sistem hakları

PUBLIC

☒ Haklarını veya rollerini tüm kullanıcılara vermeyi sağlar.

Örnek: Personel tablosunun pers\_id,ad kolonlarına webserver kullanıcısı için düzeltme yetkisi verelim;

GRANT select(pers\_id,ad)

ON Personel

TO webserver

Örnek: Personel tablosuna webserver kullanıcısı için seçme yetkisi verelim;

GRANT select

ON Personel

TO webserver

Örnek: Kazım kullanıcısı, webserver kullanıcısının personel tablosu için kayıt etme ve sorgulama yetkilerini verelim;

GRANT select,insert

ON personel

TO webserver

WITH GRANT OPTION

Örnek: Kazım kullanıcısının personel tablosunu tüm kullanıcılara sorugulama yapabilmesini sağlayalım;

GRANT select

ON kazim.personel

TO PUBLIC

Grant yetkisi ile kullanıcılara aşağıdaki tablodaki yetkiler verilebilir

| Objeler    | Table | View | Sequences | Procedure |
|------------|-------|------|-----------|-----------|
| ALTER      | ?     |      | ?         |           |
| DELETE     | ?     | ?    |           |           |
| EXECUTE    |       |      |           | ?         |
| INDEX      | ?     |      |           |           |
| INSERT     | ?     | ?    |           |           |
| REFERENCES | ?     |      |           |           |
| SELECT     | ?     | ?    | ?         |           |
| UPDATE     | ?     | ?    |           |           |

**CREATE ROLE** ☒ Kullanıcıya yetki olarak verilen rollerdir.

Yazılımı;

CREATE ROLE rol;

Örnek:

CREATE ROLE Manager;

GRANT create table, create view TO Manager;

GRANT Manager TO webserver;

Manager isminde rol tanımlandı. Tablo ve view yaratma yetkisi verildi. Bu yetkiler Webserver isminde kullanıcıya aktarıldı.

Kullanıcının şifresini değiştirmek için;

ALTER USER user\_adı IDENTIFIED BY şifre

**REVOKE** Verilen rolleri geri alır.

Yazılımı;

REVOKE rol

ON tablo

FROM kullanıcı

Örnek: webserver kullanıcısının şifre tablosuna kayıt etme,düzeltilme ve sorgulama yetkilerini kaldıralım,

REVOKE select,insert,update

ON tifle

FROM webserver

**CREATE SYNONYM** Bir objenin aynısının kopyasının alarak kendi userında yaratır.

Yazılımı;

CREATE [PUBLIC ] SYNONYM obje\_adı

FOR kullanıcı.obje

PUBLIC

☐ Tüm kullanıcıların yetkilerini alır.

Örnek: Webserver'ın personel tablosunun aynısını kendi alanımızda yaratalım

CREATE SYNONYM personel

FOR webserver.personel

**DROP SYNONYM** Bir objenin aynısının kopyasının olarak yaratılan objeyi siler.

Yazılımı;

DROP [PUBLIC ] SYNONYM obje\_adı

## ADVANCED SQL

IN dir Liste içinde belirlenen değerler

NOT IN ☐ Liste içinde olmayan tüm değerlerdir.

ANY en az birine karşılanır seçilen değerlerden

ALL hepsi karşılanır seçilen değerlerden