ORACLE SQL KOMUTLAR

29 Nisan 2021 Perşembe 10:17

ORACLE SQL KOMUTLARI

Veritabanı üzerinde yapılacak tüm işlemlerde SQL komutları kullanılacaktır. Yapılacak işlemlere örnek olarak ; veritabanında obje yaratılması, bu objelerin kullanılması, veritabanındaki tablolarında bulunan verilerin yaratılması, değiştirilmesi, silinmesi, sorgulamaların yapılması, veritabanına, tablolara erişim, kontrol ve yönetim işlevleri verilebilir. Çeşitleri isimlendirirsek ;

- **DDL (Data Definition Language) - (Veri Tanımlama Dili)**
- **DML (Data Manipulation Language) - (Veri Kullanma Dili)**
- **DQL (Data Query Language) - (Veri Sorgulama Dili)**
- **DCL (Data Control Language) - (Veri Kontrol Dili)**
- **Data administration commands - (Veri Yönetim Komutları)**
- **Transactional control commands - (İşlem Kontrol Komutları)**

1-DDL (Data Definition Language) - (Veri Tanımlama Dili)

CREATE TABLE ,
ALTER TABLE ,
DROP TABLE ,
CREATE INDEX ,
ALTER INDEX ,
DROP INDEX ,
CREATE VIEW ,
DROP VIEW
gibi Veritabanı objelerinin yaratılmasıyla ilgili komutlardır.

2-DML (Data Manipulation Language) - (Veri Kullanma Dili)

INSERT ,
UPDATE ,
DELETE
gibi en temel komutlarla veritabanı objelerini kullanma işlemleri yapılır.

Bir DML sadece Parse ve Execute safhalarını içerir.

ORACLE SQL KOMUTLARI

1 Server process, veri ve rollback bloklarını buffer cache'de bulamazsa, bu blokları veri dosyalarından okur.
2 Okunan blokları buffer cache'e kopyalar.
3 Server process veriyi kilitler.
4 Server process, yapılan değişikliği, redo log buffer içindeki rollback (önceki değer) ve veri (yeni değer) kısımlarına kaydeder.
5 Server process, database buffer cache içindeki rollback bloğuna önceki değeri kaydeder ve veri bloğunu da günceller. Buffer cache'teki her iki blok, diskteki ilgili bloklarla aynı değerlere sahip olmadıklarını belirtmesi için, dirty buffers olarak işaretlenir.

Database Buffer Cache: En son kullanılan veri bloklarını saklamak için kullanılır. Her bir buffer'ın büyüklüğü, DB_BLOCK_SIZE parametresi ile belirlenen blok boyutu ile aynıdır.

Buffer sayısı DB_BLOCK_BUFFERS ile belirlenir. Oracle server Buffer Cache'de yeni blok alanları temin etmek için Least Recently Used (LRU) algoritmasını kullanır.

3-DQL (Data Query Language) - (Veri Sorgulama Dili)

SELECT en temel komutudur.

Girilmiş olan veriler üzerinde istenilene ulaşmak amacıyla kullanılır.

ORACLE SQL KOMUTLARI

1 Parse
Bu bölümde, kullanıcı process'i sorguyu server process'e, yazım kontrolü için bir istek ile gönderir veya sorguyu işler. Server process, komutun geçerliliğini kontrol eder, ve SGA'daki Shared Pool alanını ifadeyi derlemek için kullanır. Bu safhanın sonunda server process user process'e olumlu yada olumsuz bir yanıt gönderir.
2 Execute
Bu safhada server process veriyi almak için hazırlar.
3 Fetch
Sorgu tarafından seçilen satırlar bu safhada server'dan user'a gider. Transfer için kullanılan hafızanın büyüklüğüne bağlı olarak, sorgunun sonucunu kullanıcıya aktarmak için bir yada daha çok fetch (alıp getirme) gereklidir.
Shared Pool
Parse aşamasında kullanılır. SGA'nın bir parçasıdır. İçinde bulunan Library Cache, en son kullanılan SQL ifadeleri ile ilgili bilgileri saklar. Bir ifade yeniden çalıştırılırken, eğer çalıştırma planının üzerine başka ifadelerin çalıştırma planları yazılmadıysa, server process ifadeyi yazım açısından incelemek (parse) zorunda kalmaz. Data Dictionary Cache ise, tablo ve kolon tanımlamaları, kullanıcı isimleri ve şifreler, ve ayrıcalıklar gibi en son kullanılan veri kütüphanesi bilgilerini içerir.

4-DCL (Data Control Language) - (Veri Kontrol Dili)

ALTER PASSWORD ,
GRANT ,
REVOKE ,
CREATE SYNONYM
gibi veritabanındaki objelere erişimlerin yetkilerin kontrol edilmesine yönelik komutları içerir.

5-Data Administration Commands - (Veri Yönetim Komutları)

START AUDIT ,
STOP AUDIT
gibi veritabının genel olarak performans analizine yönelik olarak sadece veritabanı yöneticisi tarafından belirli zamanlarda kullanılması gereken komutlardır.

6-Transactional Control Commands - (İşlem Kontrol Komutları)

COMMIT,
ROLLBACK,
SAVEPOINT,
SET TRANSECTION
gibi veritabanındaki tablolara kayıtların kontrol edilmesine yönelik komutlardır.

CREATE TABLE Komutu (CREATE TABLE Statement)

Bir veritabanı içerisinde, kullanıcıların veri girmeleri, değiştirmeleri ve sorgulamaları yapacakları bilgi dosyalarının yaratılması gereklidir. Bu yaratılan tablolara, sistemde yer alan tüm kullanıcılara veya belli kullanıcılara erişim yetkileri verilmelidir. Bu erişim yetkileri, dosyaya kayıt ekleme, silme, değiştirme ve kayıtları seçme yetkilerinden biri olabilir.

Veritabanını oluşturan tabloları yaratan komut CREATE TABLE dir. Bu komutla ; yaratılacak olan tablo ve bu tabloyu oluşturan alanlar, bu alanların tipleri, bu tabloya erişim özellikleri ve bu tablonun diğer tablolarla olan ilişkileri belirlenecektir.

```
CREATE TABLE tablo_adı (
      Alan_adı    veritipi,
      Alan_adı    veritipi,
      Alan_adı    veritipi,
      ...
      Alan_adı    veritipi,
      CONSTRAINT adı PRIMARY KEY (alanadı,alanadi...),
      CONSTRAINT adı FOREIGNKEY alanadi REFERENCES tablo_adi (alan_adi)
)
TABLESPACE tablespace_adi
STORAGE
(INITIAL          değer
 NEXT             değer
 MINEXTENTS       değer
 MAXETTENTS       değer
 PCTINCREASE      değer);
```

TABLO_ADI : Kullanılacak amaca uygun olarak, anlaşılabilir, türkçe karakterler ve özel karakterler içermeyen bir isim olmalıdır.
ALAN_ADI : Tablo içersinde çeşitli tipteki verilerin saklanacağı alanlara verilen isimlerdir. Burada da kullanım amacına uygun, anlaşılır ve türkçe ve özel karakterler içermeyen bir isim olmalıdır.

Kullanıcıların bu isimleri verirken daha önce dikkatini çekmeye çalıştığım özelliklere uymalarını bir kez daha hatırlatıyorum. Tablo adına bakıldığında bu tablonun ne için yaratıldığı, hangi özellikte olduğu anlaşılmalı aynı şekilde tablo içindeki alanlara verilecek olan isimlerinde barındıracağı bilgiyi açıklayıcı şekilde olmasına dikkat edilmelidir. İleri zamanlarda bu tabloya ve alan isimlerine bakıldığında hem yaratıcı hem de diğer kişilerin kolaylıkla anlamaları açısından belirli kurallara uyulması tavsiye edilmektedir. Proje bazında kullanılacak bu tablolara proje bölüm (STK, FIN, MUH vb.) takılarının eklenmesi, tablonun hangi tipte olduğunun (Parametre, Master vb.) belirtilmesi standartlarına uyulmasına dikkat ediniz.

VERİTİPİ : Alanların hangi tipte veriyi hangi uzunlukta barındıracağının belirlenmesidir. Veritipleri içinde en çok kullanılanlar ; **CHAR, VARCHAR2, NUMBER, DATE,** şeklindedir.

PRIMARY KEY : Tablo içindeki kayıtlara hızlı erişimlerde kullanılacak bir veya birden fazla alanların tanımlanması için kullanılır. En önemli özelliği bu alanlara girilecek değerlerin boş olmaması ve tekrar etmiyor olmasıdır. Bir tablo, sadece bir tek PRIMARY KEY özelliğine sahiptir. Tablolar arasındaki ilişkilendirmelerde kullanılmaktadır.

FOREIGN KEY : Bir tablo içinde belirtilen alanların başka tablolarla bağlantısının sağlandığı ifade şeklidir. Bu özellik sayesinde belirtilen alana girilecek değer, gösterilmiş olan diğer tabloda ki alana önceden girilmiş değerler arasından olmak zorundadır. Böylece verilerin güvenilirliği, doğruluğu sağlanmış olacaktır.

PRIMARY KEY ve FOREIGN KEY ile ilgili detay bilgiler ileriki sayfalarda örnekler ve açıklamaları ile verilmektedir.

TABLESPACE : Bu tablonun hangi tablespace üzerinde yaratılacağının belirlenmesi için kullanılır. Veritabanın kurulması sırasında kullanıcı tablolarının hangi tablespace üzerinde tutulacağı belirlenmiş olduğundan bu alan isminin kullanılması zorunludur. USER_DATA isminde bir tablespace tanımlaması yapılmışsa burada TABLESPACE user_data şeklinde kullanılmalıdır.

STOREGE bölümü altında ; bu yaratılan tablonun boyutunun, genişleme şartlarının ve bu tabloya erişim büyüklüklerinin değerleri tanımlanır.

```
(INITIAL          değer
 NEXT             değer
 MINEXTENTS       değer
 MAXETTENTS       değer
 PCTINCREASE      değer);
```

ALTER TABLE Komutu (ALTER TABLE Statement)

Tablo yaratıldıktan sonra unutulan bir alan veya bir bağlantı şekli veya kullanılan alanların uzunluklarının değiştirilmesi gibi özelliklerin sonrada ilave edilmesi için kullanılan komuttur. Genel olarak ALTER TABLE komutu ile yapılabilecek özellikler ;

- Alan eklenmesi,
- Tabloya ait erişim bilgilerinin eklenmesi,
- Alanların değiştirilmesi (veritipi, boyutu, default değeri),
- Tablonun yaratılması sırasında kullanılan parametrelerin değiştirilmesi,
- Tablonun kullanılmasına yönelik değerlerin değiştirilmesi,
- Tabloya ait yaratılmış erişim bilgilerinin silinmesi,

```
ALTER TABLE tablo_adi
ADD (alan_adi           veri_tipi,
     alan_adi           veri_tipi)

ALTER TABLE tablo_adi
MODIFY(alan_adi         veri_tipi,
       alan_adi         veri_tipi)
```

ADD özelliği ile tabloya yeni bir alan ilave edilmiş olacaktır.
MODIFY özelliği ile tabloda var olan bir alanın veritipi veya uzunluğu değiştirilebilir. Ancak bu özelliğin geçerli olabilmesi için tablonun boş yani hiçbir kaydın girilmemiş olması gerekir.

```
ALTER TABLE stok_model_tanim
 ADD ( CONSTRAINT stok_model_uk1  UNIQUE (TANIM));
```

Yukarıda ki örnekte stok_model_tanim dosyasına TANIM alanının UNIQUE özelliği eklenmiştir.

ALTER TABLE stok_model_tanim DROP CONSTRAINT stok_model_uk1 ;

Burada da , stok_model_tanim dosyasında yaratılmış olan stok_model_uk1 özelliği silinmektedir.

DROP TABLE Komutu (DROP TABLE Statement)

Yaratılmış olan bir tablonun silinmesi, ortadan kaldırılması amacıyla kullanılır. Tablonun yaratıldığı TABLESPACE'den kaldırılmış olacaktır.

DROP TABLE tablo_adi CASCADE CONSTRAINTS ;

Tablo içindeki bütün kayıtlarda silinmiş olacaktır bu komutun kullanılması sonrasında.

CASCADE CONSTRAINTS özelliğini kullanılmasının amacı ;
Ø Eğer silinecek tablo bir başka dosyaya FOREIGN KEY ile bağlanmışsa bu özelliği dikkate almadam tabloyu ortadan kaldıracaktır.
Ø Tablo üzerinde kurulmuş olan PRIMARY KEY veya UNIQUE KEY indexleri de silinecektir.

Ancak kullanıcıların bir tabloyu ortadan kaldırmak istediklerinde dikkat etmeleri gereken noktalar ;

Ø Ortadan kaldırılan tablonun kurtarılma şansı yoktur (backup hariç)
Ø Bu tablonun kullanıldığı VIEW, STROGE PROCEDURE, FUNCTION veya TRIGGER'lar geçersiz hale geleceğinden bunların tekrardan kontrol edilerek düzenlenmesi gerekecektir.

KISITLAMALAR (CONSTRAINTS)

Yaratılmış olan veritabanında ki tablolara geçerli ve iş kurallarına uygun şekilde verilerin girilmesi için tablo bazında, alanlar bazında veya ilgili tablolar arasında bazı kurallar konulabilir. Bu kurallar aynı zamanda kısıtlamalar şeklinde olabilir. İşte tüm bu özelliklere **KISITLAMALAR (CONSTRAINTS)** adı verilir.

Tablolara verilen bu kısıtlamalar veritabanı bazında çalışacağından tabloların alanlarına yeni kayıt girişinde veya güncelleştirmede veya silinme durumlarında devreye girecektir. Kullanıcıların yapmış oldukları programlardan bağımsız olarak çalışacağından herhangi bir atlama, unutma durumlarında yanlış, geçersiz işlemlerin yapılmasına engel olunacaktır.

Tablo ve alanlara kısıtlamaların getirilmesi veritabanın kurulması veya daha sonrasında olacaktır. Ancak bu kısıtlamaların neler olacağı önceden iş kurallarına bağlı olarak belirlenmeli ve denenmelidir. Bu tablolara ve alanlara ileride ne gibi bilgilerin girileceği ön görülerek kuralların koyulması önerilir.

Veritabanı bazında kısıtlamaların konulması bu kısıtlamaları tablo ve alanlarda kolayca izlenmesini görülmesine pek olanak sağlamaz. Veritabanında saklanan bu kısıtlamaları ancak veritabanı sistem yöneticisi (DBA) görebilir bu yüzden özelikle programların yazılması sırasında bu kısıtlamaların neler olduğunun önceden bilinmesi gereklidir. Aksi takdirde bilgi girişlerinde kısıtlamalara takılacak işlemler nedeniyle program hata verecektir. Bu da kullanıcının işlerinin aksamasına neden olur.

Kısıtlama çeşitleri aşağıdaki gibidir (Burada terimlerin türkçe karşılıkları verilmekle beraber daha kolay anlaşılması için orijinal isimleri kullanılmıştır);

- Primary Key (Öncelikli Anahtar)
- Foreign Key (Dış Anahtar)
- Unique Key (Tek anahtar)
- Column Constraint (Alan bazında kısıtlama)

Primary Key

Bir tabloya hızlı erişimde kullanılacak olan anahtar alanın veya alanların birlikte tanımlanmasıdır. Bu alan veya alanların içindeki değerler tekrar etmeyecek şekildedir. Bir anlamda tekdirler (UNIQUE).

Bir tablonun ancak ve ancak bir PRIMARY KEY olabilir. Ancak bir tablonun birden fazla UNIQUE KEY olabilir.

Primary key oluşturan alan veya alanlar boş (NULL) olamazlar. Tablonun yaratılması sırasında CREATE TABLE veya yaratıldıktan sonra ALTER TABLE ... CONSTRAINT komutlarıyla tabloya Primary key yaratılabilir.
Primary Key en fazla 16 alanın birleşmesinden oluşabilir. Ancak LONG veya LONG ROW veritiplerini içeren alanlar primary key'de kullanılamaz. Ayrıca bu erişim anahtarının toplam boyutu byte cinsinden veritabanı blok boyutunun yarısından daha az olmalıdır.

Tabloya bir primary key tanımlaması verilmişse sistem otomatik olarak bir unique key de yaratacaktır. Eğer isim verilmemişse bu sistemde SYS_Cn (n tamsayı) tanımlamasıyla yer alacaktır. Primary key veya Unique key isimlendirilmesinde de kolayca anlaşılabilir terimlerin kullanılmasına dikkat edilmelidir. Bir alan primary key olarak tanımlanacaksa NOT NULL ifadesini kullanmak gereksizdir. Çünkü bir alan primary key olarak tanımlanacaksa boş değer alamaz.

```
CREATE TABLE elemanlar (
      eleman_no   number(3) CONSTRAINT pk_satis PRIMARY KEY,
      adi         varchar2(15),
      soyadi      varchar2(10)
);
```

örneğinde olduğu gibi tablonun yaratılması sırasında alanların belirlendiği sırada tanımlama yapılabilir. Ya da

```
CREATE TABLE elemanlar (
      eleman_no   number(3),
      adi         varchar2(15),
      soyadi      varchar2(10),
      CONSTRAINT pk_satis PRIMARY KEY (eleman_no)
);
```

örneğinde olduğu gibi tablonun yaratılma komutunun en son satırında hangi alanın primary key olacağının belirtilmesi ile de yaratılabilir. Eğer tablonun yaratılmasından sonra bu tabloya primary key eklenmek istenirse ;

```
ALTER TABLE elemanlar ADD PRIMARY KEY (eleman_no) ;
```

Foreign Key

İş süreçlerine uygun şekilde tasarlanmış veritabanlarına ait tablolara giriş, düzeltme veya iptal işlemlerinde bilgilerin sürekliliği, doğruluğunun kontrol altında tutulabilmesi mutlak şarttır. Tablolara ait alanlara giriş yapılırken başka tablolarda ki alanlar kullanılabilir. Parametre tablolarına girilen değerler, master tablolarda kullanılabilir. Aynı şekilde master tablolarda kullanılan değerler detay tablolarda da aynı şekilde kullanılması gereklidir. Verinin tutarlı olması açısından son derece önemli olan bu özelliği kullanmak için Foreign Key (Yabancı tablolara erişim) yararlanalır.

Bir tabloya ait bir alan (child-sonuç), başka bir tablonun alanı (parent-esas) ile arasında Foreign Key ilişkisi kurulmuşsa artık sonuç alanına, bağlandığı tablonun esas alanındaki değerlerden başka bir değer girilmesi engellenmiş olacaktır. Aynı şekilde esas tablo alanındaki değerlerin değiştirilmesi veya silinmesi de veritabanı seviyesinde engellenmiş olacaktır. Bu kısıtlama yönteminin kullanılması ile uygulama programlarında bu kontrollerin yapılması gereksiz olacaktır.

Bir tablo içinde Foreign key kullanılması için bazı kurallara dikkat edilmesi gereklidir ;

- Esas tablo alanın Primary Key olarak tanımlanmış olması,
- Esas tablonun sonuç tablosundan önce yaratılması (CREATE TABLE) ,
- Bağlanacak alanların aynı veri tiplerinde olması,

Örneğin bir okula ait BOLUMLER tablosu (Parent), parametre tablosu olarak tanımlanmış olsun. Öğrencilerin hangi bölüme ait olduklarıda OGRENCI tablosunda bir alanda (Child) tutulacaktır. Bu OGRENCI tablosu MASTER tablo niteliğindedir.

```
CREATE TABLE bolumler (
      bolum_no    number(3),
      adi         varchar2(15),
CONSTRAINT pk_bolum PRIMARY KEY
);

CREATE TABLE ogrenci (
      Ogrenci_no  varchar2(10),
      Adi         varchar2(15),
      Soyadi      varchar2(15),
      Ogr_bolum   number(3),
CONSTRAINT fk_bolum FOREIGN KEY (ogr_bolum) REFERENCES bolumler (bolum_no)
);
```

Bu şekilde tanımlanmakta olan OGRENCI tablosunda ki ogr_bolum alanı , BOLUMLER tablosunda ki bolum_no alanı ile Foreign key kullanılarak bağlanmıştır. Böylece ogr_bolum alanına, bolum_no alanındaki değerlerden başka bir değer girilmesi mümkün değildir. Aynı şekilde bolum_no alanına girilmiş bir değer eğer ogr_bolum alanında kullanılmışsa artık bu değer değiştirilemez ve silinemez. Ancak child tablosunda kullanılması ortadan kaldırıldığında bu değer silinebilir veya değiştirilebilir.

Unique Key

Bir tablodaki bir alanın değerlerinin tekrar etmemesi için kullanılan bir kısıtlama yöntemidir. Ancak bu alanın değeri NULL olabilir. Bir tablodaki PRIMARY KEY olarak tanımlanmış alanlara UNIQUE KEY kullanılması gereksizdir zaten unique özelliğini taşımaktadır. Bir tabloda yer alan alanların arasında en fazla 16 alanın birleşmesinden unique key yaratılabilir. Ancak LONG ve LONG RAW tipli alanlarda Unique key kullanılamaz.

```
CREATE TABLE bolumler (
      bolum_no    number(3),
      adi         varchar2(15) CONSTRAINT uq_adi UNIQUE KEY
);
```

örneğinde adi alanına girilecek olan değerlerin aynı olmaması sağlanmış olacaktır. Böylece aynı bölüm isimlerinin tabloya girilmesi engellenecektir.

Column Constraint

Bir tablodaki bir alanın değerinin kontrol edilmesi amacıyla bu kısıtlama yöntemi kullanılır. İş süreçlerine uygun şekilde alanlara değerlerin girilmesinin veritabanı seviyesinde kontrol altında tutularak yanlışlıklara engel olunması sağlanmış olacaktır. Tablodaki alana uygulanacak bu kısıtlama yöntemi ancak birtanedir ve bazı kuralları vardır.

- Diğer kayıtlardaki değerlerin sorgulanarak sonuçlanması yapılamaz,
- SYSDATE, UID (UserID), USER ve USERENV gibi fonksiyonlar kısıtlamalarda kullanılamaz,
- CURRVAL, NEXTVAL, LEVEL, ROWID gibi alanlar kullanılamaz,
- Bir alanda sadece bir kısıtlama kullanılabilir

```
CREATE TABLE bolumler (
        Bolum_no       number(3) CONSTRAINT check_bolumno
                                 CHECK (bolumno BETWEEN 100 AND 999),
        Adi            varchar2(15) CONSTRAINT check_adi
                                    CHECK (adi = UPPER(adi))
);
```

Yukarıdaki örnekte "bolum_no" alanına 100 ile 999 arasında bir sayı girilmesi sağlanmış olacaktır. Ayrıca "adi" alanına girilen değeri saklarken büyük harfe çevrilecektir.

Uygulama programlarında bu kısıtlamalar haricinde değerler girilirse sistem hata mesajı verecektir ve kullanıcıların bu mesajları anlayabilmesi için programlarda bu hata mesajların kontrol edilmesi sağlanmalıdır.

DROP CONSTRAINT Komutu (DROP CONSTRAINT Statement)

Bir tablo üzerinde oluşturulmuş alan veya tablo bazlı kısıtlamaların ortadan kaldırılması için kullanılır. Ancak kullanıcıların bu komutu kullanmadan önce tablolar arasındaki bağlantıların, alanlara girilecek değerlerin kontrolünün olmayacağını unutmamalıdırlar. Özellikle çalışan veritabanlarında bu komutun kullanılmaması tavsiye edilir.

Bir tablo üzerindeki bir kısıtlamanın kaldırılmasına örnek verilmek istenirse ;

```
DROP  <tablo_ismi> CONSTRAINT <kısıtlama_adi>
```

CREATE INDEX Komutu (CREATE INDEX Statement)

Veritabanında yaratılacak tablolarda tutulacak verilere hızlı erişerek işlem yapılması esastır. Veri girişi, veri güncelleme, silme veya raporlama amaçları için tablolardaki alanlar kullanılarak erişimler yapılmaktadır. Bu erişimlerin hızlı olabilmesi için de alanların istenilen şekilde sıralı olmalıdır.

Indexlerin yaratılması sırasında iyi bir analiz yapılmalıdır. Bir tabloya yapılacak her kayıt ekleme, güncelleme, silme (INSERT, UPDATE, DELETE) işlemlerinde index tablolarının da güncelleneceği unutulmamalıdır. Bu da sistem performansını genel olarak düşürücü bir faktördür. Özellikle veriye erişim yöntemlerinin, raporlamalarda kullanılacak sorgu çeşitlerinin belirlenmesi ve hepsini kapsayacak index tabloların yaratılması önerilir. Ayrı ayrı alanlar için index yaratılacağına bu alanların birlikte kullanılacağı index tabloların yaratılmasına çalışılmalıdır. Ayrıca çalışmakta olan bir veritabanında, içinde yüklü sayıda veri olan tablolara sonradan indexlerin yaratılması sistem performansını etkileyeceği unutulmamalıdır. Ayrıca benzer alanlardan oluşan indexler varsa sorgulamalar sırasında veritabanının hangi indexi kullanacağının belirlenmesi tavsiye edilir.

Veritabanı sisteminde dahan önce gördüğümüz CREATE TABLE komutu sırasında kullandığımız PRIMARY KEY ve UNIQUE değerleri için otomatik olarak INDEX tablolar yaratılır. Böylece veriye erişmede kolaylık, hızlılık sağlanmış olacaktır.

Kullanılacak tüm SELECT ifadelerindeki WHERE kısmında yazılacak alanların INDEX tablolarında kullanılan alanlardan seçilmesi gerekir. Aynı zamanda WHERE kısmının fazla karışık olmaması tavsiye edilir. Böylece sistemin hangi index tablosunu kullanacağının belirlenmesinde zorluk yaşanmamış olur. WHERE kısmında kullanılacak alanların INDEX tablolarındaki sıraya göre yazılmaz ise INDEX kullanılmamış olacaktır. INDEX hangi sıradaki alanlar ile yaratılmışsa WHERE kısmında da o sırada kullanılması gerekmektedir.

ORACLE veritabanı, INDEX yönteminde B*-TREE denilen yöntemi kullanmaktadır. Bu yöntemde indexlenen alanlar bir ağaç yapısı şeklinde disk üzerinde tutulurlar. Indexli bir alana erişmek istenildiğinde aşağıdan yukarıya doğru bir yöntemle uygun ve istenilen verilere ulaşılmış olacaktır. INDEX tablolarında sıralanmış alanları belirleyen aıraç ROWID denilen özelliktir. ROWID her bir alanın kaçıncı kayıt olduğunu gösteren bir sistemdir. UNIQUE olan alanların INDEX değerlerinde tek bir ROWID vardır. NONUNIQUE indexlerde ise önce alanın değeri daha sonra ROWID vardır.

Sorgulamaların çalıştırılmadan önce EXPLAIN denilen yöntemle analiz edilerek doğru indexlerin kullanıp kullanmadığı araştırılmalıdır. Genel kural olarak bir sorgulama sonucunda tablonun toplam kayıt sayısının %10 ila %15'i kadar kayıt getirilecekse index yaratılması uygundur. Eğer bir sorgulama bu oranlardan daha fazla sayıda kayıt getirecekse index kullanılmasının bir yararı olmadığı gibi sistem performansını olumsuz etkileyecektir.

INDEX kullanılmadan bir tabloya erişim yapılmak istenirse, veritabanı sistemi, bu tablonun disk üzerindeki dağıtık şekilde bulunan parçacıklarına erişerek tümü bazında arama yapacaktır. Bu şekilde bir arama TABLE SCAN veya FULL TABLE SCAN olarak adlandırılır. Bu şekildeki arama, disk kafasının sürekli hareketi, sürekli bir şekilde okuma ve kontrol etme, böylece işlemci üzerindeki yükü arttıracaktır. Bütün bunlarda sistemin performansını olumsuz etkileyecektir. Veriye hızlı erişmenin tek yolu doğru bir şekilde INDEX'lenmiş tabloları doğru WHERE ifadelerini SELECT cümlecikleri içinde kullanmaktır.

```
CREATE [UNIQUE] INDEX <index_ismi>
ON {<tablo_ismi>(<alan1>,...[ASC |DESC])}
[INITRANS   sayi]
[MAXTRANS   sayi]
[TABLESPACE <tablespace_adi>]
[PCTFREE          sayi]
[NOSORT]
[RECOVERABLE | UNRECOVERABLE ]
```

Ø UNIQUE : İndekslenecek alanın tekil olacağını belirten bölümümdür.

Ø İndex ismi : Oluşturulacak indekse verilecek isimdir.
Ø Tablo ismi : İndeksin tanımlanacağı tablo ismidir.
Ø Sütun ismi : İndeksin oluşturulacağı sütunlardır.
Ø ASC : İndeksin artan olacağını belirten bölümdür.
Ø DESC : İndeksin azalan olacağını belirten böümdür.
Ø INITRANS : İndeks üzerinde aynı anda ilk olarak kaç tane işlem yapılacağının belirtildiği bölümdür.
Ø MAXTRANS : İndeks üzerinde aynı anda en fazla kaç işlem yapılacağının belirtildiği bölümdür.
Ø TABLESPACE : İndeksin hangi tablespace üzerinde oluşturulacağının belirtildiği bölümdür.
Ø PCTFREE : Sonradan yapılacak ekleme ve değiştirmeler için boş olarak ayrılacak yerin bloğa göre yüzdesinin belirtildiği kısımdır.
Ø NOSORT : Kayıtlar veritabanında sıralı olarak yer alıyorsa, indeks kayıtlarının tekrar sıralanmaması için kullanılan ifadedir.
Ø RECOVERABLE: İndeks için yapılan işlemlerin geri alma ihtimaline kaydedilmesini belirten bölümdür.
UNRECOVERABLE: İndeks için yapılan işlemlerde geri alma işlemi olmayacağının belirtildiği bölümdür.

CREATE WIEW Komutu (CREATE VIEW Statement)

Bir veritabanında yer alan tablolar ve bu tablolardaki alanların sorgulama yapılarak kullanılması ve çıkan sonuçların bir geçici dosyada (VIEW) tutulması işlemidir. Tablolar arasında bağlantılar kullanılarak sorgulamanın her seferinde yeniden yazılması yerine bir kez yazılarak sonuçların görüntü tablosunda tutulması ve bu tablo üzerinden de sorgulamaların yapılması hız, zaman ve performans kazandıracaktır.

Yaratılmış olan görüntü tablosu üzerinde INSERT, DELETE ve UPDATE işlemleri yapılamaz. Ancak görüntü tablosunu oluşturan tablolara yeni bir kayıt eklenmesi, değiştirilmesi ve silinmesi durumlarında görüntü tablosunda da güncelleştirmeler yapılacaktır.

Görüntü tabloların kullanılması, veritabanında ilgili tabloların üzerinde bağlantıların tekrar tekrar kurulması engellenmiş olacaktır. İş süreçlerine bağlı olarak özellikle ekran veya yazıcı raporların alınması, sorgulamaların gerçekleştirilmesi gibi amaçlarda kullanılır.

```
CREATE or REPLACE VIEW tablo_adi AS
      SELECT .....;
```

Bu formata uygun bir şekilde ilk defa yaratılan (CREATE) veya yaratılmış bir görüntü tablosunda değişiklik (REPLACE) yapmak mümkündür. “tablo_adi” ile oluşturulacak olan görüntü tablosuna bir isim verilecektir. Bu isimin kolayca anlaşılmasına dikkat edilmelidir. Tabloyu oluşturacak sorgulama SELECT ifadesinin içeriğinde yer alacaktır.

Görüntü dosyasının oluşturulması sırasında yeni bir alan kullanılabilir bu takdirde bu alana bir isim verilmesi gereklidir.

Görüntü dosyasının yaratılması sırasında ORDER BY ile sıralama yapılması özelliği kullanılmaz. Buna karşılık GROUP BY özelliği kullanılabilir.

Örnek olarak aşağıdaki BOLUMLER ve OGRENCILER tablolarının “bolum_no” ile bağlanarak ogrenci nin adı ve soyadı ile bölüm adının listeleneceği bir görüntü dosyası oluşturalım.

```
CREATE TABLE bolumler (
      bolum_no    number(3),
      adi         varchar2(15),
CONSTRAINT pk_bolum PRIMARY KEY
);

CREATE TABLE ogrenci (
      Ogrenci_no  varchar2(10),
      Adi         varchar2(15),
      Soyadi      varchar2(15),
      Ogr_bolum   number(3),
CONSTRAINT fk_bolum FOREIGN KEY (ogr_bolum) REFERENCES bolumler (bolum_no)
);

CREATE or REPLACE view_ogr_bolum AS
SELECT a.adi||' '||a.soyadi ISIM, b.adi BOLUM
  FROM ogrenci a, bolumler b
 WHERE a.ogr_bolum = b.bolum_no;
```

Bu örnekte OGRENCI tablosundaki "adi" ve "soyadi" alanları birleştirilerek "ISIM" alanı olarak, BOLUMLER tablosundaki bolum adi "adi" alanı ise "BOLUM" alanı olarak kullanılmıştır. Yaratılan view_ogr_bolum tablosunda sorgulama yapılmak istenirse,

```
SELECT *
  FROM view_ogr_bolum
 ORDER BY isim;
```

DROP VIEW Komutu (DROP VIEW Statement)

Yaratılmış bir görüntü tablosunun veritabanından silinmesi için kullanılır.

DROP VIEW <tablo_adi> şeklinde kullanılır. Özellikle yaratılmış ve programlar tarafından kullanılmakta olan bir görüntü tablosunun silinmesi durumunda programların çalışmayacağı unutulmamalıdır.

CREATE SEQUENCE Komutu (CREATE SEQUENCE Statement)

Bir sayaç tablosu olarak isimlendirebileceğimiz "Sequence" tipi tablolar verilen bir sayıdan başlayarak artan veya azalan değerlerde sayı üretilmesi için kullanılır. Özellikle başka tablolarda yer alan ve kullanıcı müdahalesine gerek kalmadan programcılar tarafından sisteme yüklenen numara alma işlemleri için kullanılır.

```
CREATE SEQUENCE <sayac_tablo_ismi>
[START WITH       sayi]
[INCREMENT BY     sayi]
[MINVALUE          sayi | NOMINVALUE]
[MAXVALUE          sayi | NOMAXVALUE]
[CYCLE | NOCYCLE ]
[CACHE            sayi | NOCACHE]
[ORDER |  NOORDER]
START WITH                Sayacın başlayacağı değerdir.
INCREMENT BY             Sayacın kaçar kaçar artacağını gösteren değerdir.
```

MINVALUE Sayacın alacağı en ufak değeri gösterir.
NOMINVALUE İlk değer olarak seçilidir. Sayaçların artması durumunda değeri 1, azalması durumunda -
değerini alır.
MAXVALUE Sayacın alacağı en büyük değeri gösterir.
NOMAXVALUE İlk değer olarak seçilidir. Sayaçların artması durumunda ki değeri , azalaması durumunda –1 dir.
CYCLE Sayaç almanın ilk veya son değerine ulaştığında tekrardan sayı üretmeye başlayacağını
gösterir.
NOCYCLE İlk değer olup, sayaç almanın ilk veya son değerine ulaştığında sayı üretilmesi durur.
CACHE Sayaç alınması sırasında kaç sayının bellekte tutulacağını gösterir.
NOCACHE Türetilen sayılar bellekte tutulmayacak anlamındadır.
ORDER Sayaç tablosunun üreteceği sayılar sıralı olacaktır.
NOORDER Sayaç tablosu rastgele sırada sayı üretecektir.

Aşağıdaki örnekte “sayac_al” tablosu 1’den başlayarak her defasında 1 artarak sayı üretecektir.

```
CREATE SEQUENCE sayac_al
      START WITH 1
      INCREMENT BY 1
      ORDER;
```

Bu “sayac_al” tablosundan yeni bir sayı alınması için NEXTVAL deyimi kullanılır;

```
SELECT sayac_al.NEXTVAL FROM dual;
```

“sayac_al” tablosunun şuanki değerini görmek için CURRVAL deyimi kullanılır;

```
SELECT sayac_al.CURRVAL FROM dual;
```

Bu tür işlemlerin yapılabilmesi içi DUAL tablosundan yararlanılır.

DROP SEQUENCE Komutu (DROP SEQUENCE Statement)

Oluşturulmuş bir sayaç alma tablosunun veritabanından silinmesi, düşürülmesi amacıyla kullanılır.

```
DROP SEQUENCE <sayac_tablo_ismi>
```

SELECT Komutu (SELECT Statement)

SELECT ifadesi veri tabanındaki bir veya daha fazla tablodan verileri alır ve sorgu sonucu olarak getirir.

SELECT ifadesinin genel yazılımı aşağıdaki gibidir.

```
SELECT [ALL | DISTINCT] seçim_listesi (select_list)
[FROM {tablo_adı(table_name)|görüntü_adı(view_name)}
```

[ipuçları (optimizer_hints)]]
[WHERE cümleciği (WHERE clause)]
[GROUP BY cümleciği (GROUP BY clause)]
[HAVING cümleciği (HAVING clause)]
[ORDER BY cümleciği (ORDER BY clause)]

Yukarıdaki ifadelerde yer alan cümleciklerin açıklamaları :

ALL

Sorgulama sonucunda yer alacak bütün satırları getirir. Tekrarlayan satırları tekrar adedi kadar getirir.

DISTINCT

Sorgulama sonucunda tekrar eden satırları bir kere getirir. Varsayılan değer (default) ALL dur.

seçim_listesi (select_list)

Tablo(lar)da seçilecek alanların (field) isimlerinin yazıldığı yerdir.
Aşağıdakilerden biri veya birkaçı olabilir.

- Asterisk (*), FROM cümleciğinde seçilen tablolardaki kolonların hepsini yaratma sırasına göre getirir.
- Görülmek istenen sıra ile yazılmış kolon isimlerinin listesi. Birden fazla alan adı varsa virgülle ayrılmalıdırlar.
- Alan adı ve alan başlığı ile birlikte yazılması. Alan adı içinde boşluk varsa ' ' arasına yazılmalıdır.
Örneğin:
SELECT CREDATE 'YARATMA TARIHI' FROM deneme
- Bir ifade olabilir. Bu ifade alan adı, sabit, fonksiyon, veya alanların çeşitli kombinasyonları veya altsorgu (subquery) olabilir.
- Lokal veya global değişken.
- Birden fazla tablo kullanılıyor ve bu tablolarda aynı alan isimleri varsa tablo adı veya sembolleri ile birlikte yazılır.

FROM {tablo_adı(table_name)|görüntü_adı(view_name)}

Bu kısıma verilerin alınacağı tablo(lar) veya görüntülerin adları yazılır.
Seçim listesinde bulunan kolonların ait olduğu tabloların adları FROM ifadesinde bulunmalıdır.
Aksi halde "geçersiz kolon adı" hatası görülür. Birden fazla tablo kullanılıyorsa bunları sembollerle ifade etmekte mümkündür.

SELECT a.credate, a.creuser, b.credate, b.creuser FROM fatura a, fatura_detay b

WHERE cümleciği (WHERE clause)

Bu bölümde şeçim yapılacak tablolar arası bağlantılar kurulur (join) ve tabloların çeşitli alanlarına istenen kriterler verilir.

- Mantıksal kontrol
- Aralık kontrol (BETWEEN)
- Sahip olma kontrol (IN)
- İçinde yer alma kontrolü (LIKE)
- Boş değer kontrolü (NULL)

Mantıksal Kontrol

=,<>,<,<=,>,>= işaretlerini kullanılarak yapılan karşılaştırmadır.

Eğer karşılaştırma sonucu doğru (true) ise TRUE değeri üretilir.
Eğer karşılaştırma sonucu yanlış (false) ise FALSE değeri üretilir.
Eğer karşılaştırılan alanlardan herhangi biri NULL ise test sonucu NULL değeri üretilir.

Aralık Kontrol (BETWEEN)

Kayıtların verilen aralıkta olup olmadığını kontrol eder. Verilen aralığın sınırları da kontrole dahildir.
Örneğin,

```
SELECT * FROM sehirler
WHERE ilkodu BETWEEN 34 AND 45
```

Sahip olma kontrol (IN)

Kayıtların verilen bir listede olup olmadığı test edilir.
Örneğin,

```
SELECT * FROM sehirler WHERE ilkodu [NOT] IN (34,35,6)
```

Burada NOT ifadesi ile karşılaştırmanın tersini de yapmak mümkündür. Yukarıdaki cümlecikte kullanılırsa
ilkodu 34,35,6 olmayan iller listelenmiş olacaktır.

IN içinde yer alacak değerler sayısal veya karakter tipinde olabilir.

Benzer kontrolü (LIKE)

Karşılaştırma yapılacak değerin alan içinde yer alıp alınmadığının kontrolüdür.

```
SELECT * FROM sehirler WHERE iladi LIKE 'A%'
```
(iladı alanında A ile başlayanlari getirir. Adana, Ankara gibi)

```
SELECT * FROM sehirler WHERE iladi LIKE '%A%'
```
(iladı alanında A olanları getirir. Adana, Ankara, Van, Manisa gibi)

```
SELECT * FROM sehirler WHERE iladi LIKE '_a%'
```
(iladı alanında ilk karakteri ne olursa olsun ikinci karakteri A olanları getirir. Manisa, Van gibi)

"_" : Kullanıldığı yere göre tek bir karakterin yerini tutar.
"%" : Kullanıldığı yerin gerisinde veya ilerisindeki karakterleri kapsar.

Boş değer kontrolü (IS NULL)

Boş alanların seçimi için kullanılır.

Örneğin;

```
      SELECT * FROM sehirler  WHERE iladi IS NULL
```

Sehirler tablosunda iladi girilmemis (boş olan) kayıtlar listelenir.

```
      SELECT * FROM sehirler  WHERE iladi IS NOT NULL
```

Sehirler tablosunda iladi girilmis (boş olmayan) kayıtlar listelenir.

İçinde yer alma kontrolü (IN)

Karşılaştırılan alanın değeri verilen ifadelerin içinde olanların seçilmesi için kullanılır.

Örneğin;

```
      SELECT * FROM sehirler  WHERE iladi IN (‘ISTANBUL, ‘ANKARA)
```

Sehirler tablosunda iladi ISTANBUL ve ANKARA’ya eşit olanlar kayıtlar listlenir.

```
      SELECT * FROM sehirler  WHERE iladi NOT IN (‘ISTANBUL, ‘ANKARA)
```

Sehirler tablosunda iladi ISTANBUL ve ANKARA dışında kalan kayıtlar listelenir.

WHERE cümleciği içinde yer alan bu kontrolleri AND , OR kullanarak birleştirmek mümkündür.

ORDER BY Cümleciği

Sorgulama sırasının belirlendiği ifadedir. Default (temel) değeri ASC olup küçükten büyüğe doğrudur. İfadenin sonunda DESC kullanılırsa büyükten küçüğe doğru sıralama yapılacak anlamındadır.

Örneğin;

```
      SELECT * FROM sehirler  ORDER BY iladi
```

Sehirler tablosundaki tüm kayıtlar “iladı” alanı sırasında küçükten büyüğe doğru sıralanır.

```
      SELECT iladi, ilkodu FROM sehirler ORDER BY 2
```

Sehirler tablosundaki tüm kayıtlar SELECT cümleciğinde yer alan alan adlarından 2.si olan ilkodu alanına göre küçükten büyüğe doğru sıralanır.

GROUP BY Cümleciği

Bir tabloda yer alan alanların gruplandırılarak istenilen SQL fonksiyonlarının kullanılması içindir.

ORDERNO	LINENO	PARTNO	QUANTITY
9001	1	101	15
9001	2	102	10
9002	1	101	25
9002	2	103	50
9003	1	101	15
9004	1	102	10
9004	2	103	20

Örneğinde LINEITEMS tablosunda, LINENO alanına göre QUANTITY toplamlarını almak

istediğimizde

```
SELECT lineno, SUM(quantity)
  FROM lineitems
  GROUP BY lineno;
```

İfadesinin sonucu

```
LINENO     SUM(QUANTITY)
---------- -------------
1           65
2           80
```

GROUP BY ifadesinde yer alacak alanlarla SELECT satırında yer alacak alanların aynı sayıda ve sırada olması gereklidir. Eğer SELECT satırında kullanılan bir alan, GROUP BY satırında kullanılmaz ise gruplandırma yapılamayacaktır.

HAVING Cümleciği

GROUP BY ifadesi ile kullanılan fonksiyonun kontrolü yapılmak istenirse HAVING cümleciği kullanılır. SELECT satırında yer alan hesaplama sonucunda bulunan değerin kontrol edilip ondan sonra gösterilmesi içindir. Yukarıda ki örnekte toplamın 70'den büyük olanların gösterilmesi istenirse o zaman ;

```
SELECT lineno, SUM(quantity)
  FROM lineitems
  GROUP BY lineno
  HAVING SUM(quantity)>70;
```

İfadesinin sonucu

```
LINENO     SUM(QUANTITY)
---------- -------------
2           80
```

böylece SQL ifadesi gruplandırmayı LINENO alanına göre yapar ve bulduğu sonucu HAVING satırında ki değerle kontrol eder, 70'den büyükse ekrana getirir aksi takdirde ekrana getirmeyecektir.

Bir tabloda gruplandırma yapılırken kullanılan alanın veya alanların istenilenler arasından seçilmesi için WHERE cümleciğinin kullanılması gerekir. WHERE, kayıt (record) sırasında seçimler için kullanılır, HAVING ise SQL'ün çalışması sonucu bulunan değerin seçiminde yani sonucun kontrol edilmesinde kullanılır.

SELECT içinde ROWID Kullanılması

SELECT komutunu kullanırken tablonun, içindeki kayıtların giriş şeklinde göre numaralandırılması ve istenilen kayıt sayısı kadar listelenmesi için kullanılan bir özelliktir.

Bu örnekte ogrenci tablosunun ilk 10 kaydı listenelecektir.

```
SELECT rowid, adi FROM ogrenci WHERE rowid<10
```

SQL GURUP FONKSİYONLARI

Bir Alanın Toplamını Hesaplama (SUM)

SUM(kolon_adı) fonksiyonu, bir alandaki verilerin toplamını hesaplar. Kolondaki verilerin hepsi sayısal veri tipinde(data type) (tam sayı(integer), ondalıklı sayı(decimal),devirli sayı(floating point), para(money)) olmalıdırlar.

Bir Kolonun Averajını Hesaplama (AVG)

AVG(kolon_adı) fonksiyonu, bir kolondaki verilerin averajı(aritmetik ortalaması)nı hesaplar. Kolondaki verilerin hepsi aynı sayısal (numeric) veri tipinde (data type) (tam sayı(integer), ondalıklı sayı(decimal), devirli sayı(floating point),para(money)) olmalıdırlar.

En büyük ve En küçük Değerlerin Bulunması (MIN ve MAX)

MIN(kolon_adı) ve MAX(kolon_adı) fonksiyonları sırasıyla, bir kolondaki en küçük ve en büyük değeri bulurlar. Kolondaki veriler sayısal (numeric), zaman (date/time), karakter (string) veri tipinde olabilirler. MIN ve MAX fonksiyonlarının sonuçları da kendilerini oluşturan kolondaki verilerle aynı veri tipindedirler.

Verileri Sayma (COUNT)

COUNT(kolon_adı) fonksiyonu, karşılaştırma değerine uyan kayıtların sayısını verir. Sonuç tam sayıdır. Alanı oluşturan veriler arasında boş (NULL) değerler varsa bunlar sayılmazlar.

```
CREATE TABLE satis (
      satis_no    varchar2(10),
      tarih       date(15),
      tutar       number(15),
);

Satis_no    Tarih       Tutar
--------    ----------  ---------
10001       01/10/2001  10000000
10002       01/10/2001  15000000
10003       10/10/2001  5000000

SELECT SUM(tutar) “Satış Toplamı”
  FROM satis

Satış Toplamı
-------------
30000000
SELECT MAX(tutar) “En büyük Satış”
  FROM satis

En büyük Satış
--------------
15000000
```

```
SELECT MIN(tutar) “En küçük Satış”
  FROM satis

En küçük Satış
--------------
5000000
SELECT AVG(tutar) “Ortalama Satış”
  FROM satis

Ortalama Satış
--------------
10000000
SELECT count(*) “Satış Sayısı”
  FROM satis

Satış Sayısı
------------
3
SELECT tarih,sum(tutar) “Toplam” ,count(*) “Adet”
  FROM satis

GROUP BY tarih

Tarih       Toplam    Adet
----------  --------  ----
01/10/2001  25000000  2
10/10/2001  5000000   1
```

DUAL TABLOSU

Bazı SQL ifadelerinin kullanılması ve denenmesi için ORACLE veritabanında DUAL isimli bir tablo yer almaktadır.Sistem tarafından yaratılan bu tabloya tüm kullanıcılar erişebilmektedir. DUMMY isimli , VARCHAR2(1) veri tipinden oluşan bir tablodur. Tek bir kayıttan oluşan bu tabloda kullanıcıların SELECT ifadesi içersinde çeşitli özellikleri kullanmalarına olanak tanır.

Bizde aşağıdaki SQL fonksiyonlarını örneklerle anlatmak istediğimizde bu DUAL tablosundan yararlanacağız.

SQL FONKSİYONLARI

Bu bölümde SQL ifadeleri içinde sıkça kullanılan bazı SQL fonksiyonları anlatılacaktır. Kullanılabilecek tüm fonksiyonlar ilgili dökümanlarda bulunacağı unutulmamalıdır. Bu bölüme alınan fonksiyonlar kullanıcıların işine yarabilecekleri fonksiyonlardan seçilmiştir. Ayrıca bazı fonksiyonların en çok kullanılan özellikleri gösterilmektedir.

Karakter Fonksiyonları

CONCATE(karakter1,karakter2,...)

Karakter biçimde verilen değerleri ard arda birleştiren fonksiyondur. || şeklinde de kullanılabilir.

Aşağıdaki örnekte ogrenciler tablosundaki kayıtlar, “adi” ve “soyadi” alanları

birleştirilerek listelenecektir.

```
SELECT adi||’ ‘||soyadi ‘Ogrenci’ FROM ogrenciler
```

LOWER(karakter)

“karakter” içersinde büyük harfle yazılmış ifadeyi küçük harfli haline dönüştürür.

UPPER(karakter)

“karakter” içersinde küçük harfle yazılmış ifadeyi büyük harfli haline dönüştürür.

REPLACE(karakter,aranacak_karakter,yerleştirilecek_karakter)

“karakter” içersinde , “aranacak_karakter” ile verilen ifadeyi arar ve bulduklarının yerine “yerleştirilecek_karakter’ ile verilen ifadelerle değiştirir.

```
SELECT REPLACE('JACK and JUE','J','BL') "Changes"
     FROM DUAL;

Changes
--------------
BLACK and BLUE
```

SUBSTR(karakter,m,n)

“karakter” içersinde , m sayısı ile verilen yerden başlayarak n sayısı kadar karakteri alır.

LENGTH(karakter)

“karakter” ifadesinin uzunluğunu sayı değerinde verir.

Tarih Fonksiyonları

SYSDATE

Sistemin o andaki tarih ve saat bilgisini verir.

```
SELECT TO_CHAR (SYSDATE, 'DD/MM/YYYY HH24:MI:SS') "Şimdiki zaman” FROM DUAL;

Şimdiki zaman
-------------------
18/11/2001 18:05:37
```

NEXT_DAY(tarih,ifade)

Verilmiş olan “tarih”e bağlı olarak “ifade” yer alan gün bilgisinin eklenerek hangi tarih olduğunun bilgisini verir. Aşağıdaki örnekte 18/11/2001 tarihine bağlı olarak önümüzdeki Saturday (Cumartesi) günün hangi tarihe geldiğini verecektir.

```
SELECT NEXT_DAY(TO_DATE('18-11-2001','dd-mm-yyyy'),'Saturday') “Gün” FROM DUAL;
```

```
Gün
--------------
24/11/2001
```

LAST_DAY(tarih)

Verilen “tarih” bilgisinin ait olduğu ayın son gününü verir.

```
SELECT LAST_DAY(TO_DATE('17-09-1992','dd-mm-yyyy')) “Ayın Son Günü” FROM DUAL;

Ayın Son günü
-------------
30-09-1992
```

ADD_MONTHS(tarih,sayı)

Verilen “tarih” bilgisine “sayı” ekleyerek hangi ayın hangi günü olduğu bilgisini verir.
17-09-1992 tarih bilgisine 2 ay eklendiğinde hangi tarih olduğunu verir.

```
SELECT ADD_MONTHS(TO_DATE('17-09-1992','dd-mm-yyyy'),2) “Tarih” FROM DUAL;

Tarih
-------------
30-11-1992
```

TRUNC(tarih,format)

Bu fonkisyon aşağıdaki tabloda gösterilen “format” şekillerine göre verilmiş olan “tarih” bilgisini dönüştürür.
“tarih” bilgisi o anki sistem tarihi (15/06/2001) olarak kabul edilirse ;

```
YYYY,YEAR
Girilen tarihe göre yılın ilk gününü verir.
SELECT trunc(sysdate,’YYYY’) from dual;

trunc(
-----------
01-JAN-2001
Q
Girilen tarihe göre en yakın çeyrek ayın ilk gününü verir. OCAK-ŞUBAT-MART ilk çeyrek,
NİSAN-MAYIS-HAZİRAN ikinci çeyrek dilimlerdir.
SELECT trunc(sysdate,’Q’) from dual;

trunc(
-----------
01-APR-2001
MON,MONTH,MM
Girilen tarihin ait olduğu ayın ilk gününü verir.
SELECT trunc(sysdate,’MON’) from dual;

trunc(
-----------
01-JUN-2001
```

DY
Girilen tarihin ait olduğu haftanın Pazar gününün tarihini verir.

W
Giirlen tarihe göre önceki haftanın Perşembe günün tarihini verir.

Dönüştürme Fonksiyonları

TO_CHAR(‘tarih’,format)

‘tarih’ biçiminde verilen değeri, ‘format’ ile belirtilen şekile çeviren fonksiyondur.

D
Haftanın gün sayısını verir. (Pazar 1, Pztesi 2...)
To_char(sysdate,’D’)
DD
Tarihin gün değerini ay içindeki sayısı (1..31)
To_char(sysdate,’DD’)
DAY
Tarihin gün adını verir. (SUNDAY,MONDAY...)
To_char(sysdate,’DAY’)
DDD
Tarihin gün değerini yıl içersindeki sayısı (1..366)
To_char(sysdate,’DDD’)
HH,HH12
Saat değerini AM,PM olarak verir
To_char(sysdate,’HH12’)
HH24
Saat değerini 0..23 arasında verir
To_char(sysdate,’HH24’)
MI
Dakika değerini verir.
To_char(sysdate,’MI’)
MM
Tarihin ay sayısını verir (01..12, Ocak 01)
To_char(sysdate,’MM’)
MON
Tarihin ay değerinin kısa adını verir
To_char(sysdate,’MON’)
MONTH
Tarihin ay değerinin uzun adını verir
To_char(sysdate,’MONTH’)
Q
Tarihin hangi çeyrek içersinde olduğunu verir (1..4)
To_char(sysdate,’Q’)
WW
Tarihe göre yıl içinde kaçıncı haftaya ait olduğunu verir.(1..53)

W
Tarihe göre ay içinde kaçıncı haftaya ait olduğunu verir (1..5)

YY,YYYY
Tarihin yıl değerini verir (2 hane veya 4 hane olarak)
To_char(sysdate,’YY’)
YEAR
Tarihin yıl değerinin ingilizce yazılış şeklini verir.

To_char(to_date(’18/11/2001’,’DD,MON,YY’)
18,NOV,01

To_char(sysdate,’DD/MM/YYYY HH24:MI)
06/01/2001 16:30

To_char(sysdate,’Month’)
Ocak

TO_CHAR(sayi,format)

‘Sayi’ biçiminde verilen değeri, ‘format’ ile belirtilen şekile çeviren fonksiyondur.

9
Sayı değerindeki hane sayısı kadar gösterim şekli
To_char(1234,’9999’)
G
Sayı değerini guruplandırır şekilde gösterir
To_char(12345,’999G999’)
D
Sayı değerini noktalı şekilde gösterir
To_char(‘123.254,’999D999’)
L
Sayı değerini ulasal para formatında gösterir (TL, $ gibi)
To_char(‘123254,’999G999L’)
S
Sayı değerini + veya – olarak gösterir
To_char(‘-125’,’999S’)

To_date(karakter,format)

‘Karakter’ biçiminde verilen değeri, ‘format’ olarak gösterilen şekilde tarih biçimine çevirir.

To_date(‘18112001’,’DD/MM/YYYY’)
18/11/2001
To_date(‘27032001,’DD-MM-YYYY’)
27-03-2001

INSERT Komutu (INSERT Statement)

Bir tabloya yeni bir kayıt eklenmesi için kullanılan SQL ifadesidir.

INSERT INTO table_adi (alan1, alan2,...,alann) VALUES (deger1,deger2,...,degern) ;

INSERT ifadesinde alanların sırasıyla VALUES içinde verilen deger'lerin sırası aynı olmalı ve alanların ozellikleri ile aynı ozellikte degerler kulllanılmalıdır. Bu ifade de istenilen alanlar kullanılarak yeni bir kaydın eklenmesi mümkün olur.

Ø Tablonun yaratılması sırasında NOT NULL değeri verilmiş bir alan kullanılmışsa mutlaka INSERT ifadesinde yer almalıdır.

Ø Tablonun yaratılması sırasında DEFAULT değeri kullanılan bir alan varsa bu alan INSERT ifadesinde kullanılmayabilir. Tabii değeri DEFAULT değerinden farklı bir değerse kullanılmalıdır.

```
CREATE TABLE ogrenci (
      ogrenci_no        NUMBER(8) NOT NULL,
      ogrenci_adi       VARCHAR2(10),
      ogrenci_soyadi    VARCHAR2(10),
      ogrenci_dog_tar   date,
      ogrenci_donem     NUMBER(4),
      ogrenci_harc      NUMBER(15),
      cre_date          date DEFAULT sysdate
);
```

örnek tablosuna kayıt eklenecekse mutlaka ogrenci_no alanın girilmesi gereklidir.

```
INSERT INTO ogrenci (ogrenci_no, ogrenci_adi, ogrenci_soyadi)
 VALUES (20101001,'AHMET','OZKAN');
```

Dikkat edilirse bu örnekte ogrenci_dog_tar alanına deger girilmemiştir. Bu kayıtta bu alan boş (NULL) olarak yer alacaktır. Cre_date alanın değeri ise kaydın yaratıldığı andaki tarih değerini otomatik olarak alacaktır.

```
INSERT INTO ogrenci (ogrenci_adi, ogrenci_soyadi,ogrenci_dog_tar,ogrenci_no)
 VALUES ('AHMET','OZKAN',to_date('01/09/2001','DD/MM/YYYY',20101001);
```

İfadesi de doğrudur. Tek fark, alanların kullanım sırasıdır. Ama alanlara karşı verilen değerler aynı sıradadır.

```
INSERT INTO ogrenci (ogrenci_adi, ogrenci_soyadi,ogrenci_dog_tar,ogrenci_no)
 VALUES ('AHMET',0,01/09/2001);
```

İfadesi ise yanlıştır ve SQL cümleceği çalışmayacaktır.. Ogrenci_soyadi alanı alfasayısal bir değer olmasına karşılık sayısal bir değer verilmiştir. Ogrenci_dog_tar alanı tarih formatında olmasına rağmen uygun bir formatta olmayan bir değer girilmiştir. Son olarak ogrenci_no alanı kullanılmasına rağmen bu alana bir değer ataması yapılmamıştır.

UPDATE Komutu (UPDATE Statement)

Bir tabloda yer alan bir veya birden fazla alanın değerlerinin değiştirilmesi için kullanılan SQL ifadesidir. Bu alanlar bir kayıta ait olabileceği gibi birden fazla kayda da ait olabilir. Bunun için genellikle UPDATE cümleciğinde WHERE ifadesi de kullanılır.

```
UPDATE table_adi SET alan1=deger1, alan2=deger2... alann=degern [WHERE ....]
```

Eğer bir UPDATE cümlesinde WHERE ifadesi kullanılmamışsa bu değişiklik bütün tablo üzerinde olacak demektir. Özellikle bu durumlarda kullanıcıların dikkat etmeleri gerekmektedir. WHERE cümleciğinin kullanılmadığı veya yanlış kullanıldığı durumlarda bütün tablonun

kayıtlarının güncelleştirileceği göz önünde bulundurulmalıdır.

Yukarıda ki OGRENCI tablosunda girilmiş olan kayıtların içersinde ogrenci_donem 2000 ve ustu olanların ogrenci_harc alanın değerini 20000000 yapmak istersek ;

```
UPDATE ogrenci
   SET ogrenci_harc=20000000
 WHERE ogrenci_donem>=2000 ;
```

İfadesini yazmamız gerekecektir.

DELETE Komutu (DELETE Statement)

Bir tablonun istenilen kaydının veya kayıtlarının veya bütün tablonun içeriğinin silinmesi istendiğinde kullanılan SQL cümleciğidir. Tablonun bütün kayıtlarının silinmesi, tablonun boyutunu , bir başka deyişle , yer aldığı TABLESPACE'de kapladığı alanı küçültmez. Tablonun kapladığı alan ne kadarsa bu değer değişmeyecektir. DROP komutuyla karıştırılmamalıdır. Eğer DELETE komutunda WHERE ifadesi yer almaz ise tabloda ki bütün kayıtlar silinecek anlamındadır.

```
DELETE tablo_adi [WHERE ....];
```

Yukarıda ki OGRENCI tablosunda girilmiş olan kayıtların içersinde ogrenci_donem 2000 altında olan kayıtları silmek istersek ;

```
DELETE ogrenci WHERE ogrenci_donem<2000 ;
```

İfadesini yazmamız gerekecektir.

ROLLBACK ve COMMIT Komutları (ROLLBACK - COMMIT Statement)

Yukarıda ki INSERT, UPDATE ve DELETE komutlarının çalıştırılması sadece çalıştıran kullanıcıyı etkileyecektir. Veritabanını kullanan diğer kullanıcılar bu işlemlerden etkilenmezler. Diğer kullanıcılarında bu işlemleri görmelerini istersek o takdirde bu komutlardan sonra COMMIT komutunu kullanmamız gerekecektir. Böylece tablolar üzerinde yapılan işlemler geçerli olacak ve diğer kullanıcılarda bunları göreceklerdir.

1 Server process, redo log buffer'a bir commit kaydı yapar. Bu kaydı yaparken SCN(System Change Number) kullanır. Bir transaction commit edildiğinde, Oracle Server bu transaction'a, veritabanı için unique olan ve devamlı artan bir SCN verir.
2 LGWR redo log buffer girdilerini, commit kaydı ile beraber, redo log dosyalarına yazar. Bu noktadan sonra, dosya okuma hatası oluşmadıkça, Oracle server değişiklerin kaybolmayacağını garanti eder.
3 COMMIT'in tamamlandığı kullanıcıya bildirilir.
4 Server process, transaction'ın bittiğine ve kilitlerin kaldırılabileceğine dair bilgiyi kaydeder.

COMMIT'in avantajları:

Log dosyalarına sıralı yazmak, veri dosyalarındaki farklı bloklara yazmaktan daha hızlıdır. Log dosyalarına yazılan değişiklikleri kaydeden bilgi çok küçüktür. Bunun yanında veri dosyalarına yazmak, verinin tüm bloklarının yazılmasını gerektirir. Redo log

buffer kısmen dolana kadar, transaction başına sadece bir eşzamanlı yazma gereklidir (transaction başına birden az olabilir). Transaction'ın büyüklüğü COMMIT işleminin harcadığı zamanı etkilemez.

Ancak herhangi bir işlemden sonra geri dönülmek istendiğinde ROLLBACK komutunu kullanmak gereklidir. Bu durumda kullanıcının yapmış olduğu işlemler geri alınarak tablo eski haline döndürülür. Veritabanında özellikle tablolarda yapılan değişiklikler çok sayıda kaydı kapsıyorsa önceki halleri ROLLBACK TABLESPACE'de tutulurlar.

Ø COMMIT komutunu kullandıktan sonra geri dönüş yoktur bundan dolayı ROLLBACK komutu kullanılamaz.

TABLOLAR ARASI BAĞLANTI (JOIN)

İki veya daha fazla tablo arasında bağlantı (join) kurmak demek veritabanına tabloların hangi alanlarının birbirleri ile alakalı olduğunu göstermek demektir.

```
WHERE a.ILKODU = b.ILKODU
```

örneğinde seçim yapılan tabloların ilkodu alanlarının birbirleriyle bağlantılı olduğu anlatılmaktadır. Eğer bu bağlantı kurulmazsa sorgu sonucunda seçim listesine yazılan tabloların kayıt sayısının kartezyen çarpımı kadar kayıt gelir. Örneğin yukarıdaki bağlantı kurulmamış olsa idi listeye "a" tablosu ile "b" tablosunun kayıt sayısının çarpımı kadar kayıt listelenecekti.Yani "a" tablosundaki her kaydı "b" tablosundaki bütün kayıtlarla eşleştirme yaparak listeleyecektir.

Tablolar arası bağlantıları ikiye ayırabiliriz.

İçine Bağlantı (Inner Join)
Dışına Bağlantı (Outer Join)

İçine Bağlantı (Inner Join)

İçine bağlantılarda listeye sadece bağlantı kurulan alanları eşleşebilen kayıtlar gelir.İçine bağlantı, bağlantı kurulan alanlar arasına "=" işareti konarak yazılır. Bizlerin bugüne kadar öğrendikleri bu şekilde bağlantıydı.

Örneğin

```
WHERE a.ILKODU = b.ILKODU
```

Bu örnekte "a" tablosunda ilkodu boş(null) olmayan ve de ilkodu "b" tablosunda bulunan kayıtlar listelenir. "a" tablosunda ilkodu boş(null) olanlanlara karşılık "b" tablosunda bir eşleşme olamayacaktır, çünkü null=5 gibi bir eşleşme olamaz. Ayrıca "b" tablosunda olmayan bir ilkoduna sahip firmalar da listelenemezler, çünkü ilkodu 15 olan bir firma varsa ve "b" tablosunda da 15 kodlu il yoksa eşleşme olamaz.

Dışına Bağlantı (Outer Join)

Dışına bağlantılarda listeye tabloların birindeki tüm kayıtlar ve diğerinde ise bağlantıya karşılık gelen kayıtlar gelir.
Dışına bağlantı, bağlantı kurulan alanlar arasına "=" işareti ve tüm kayıtların getirilmesi istenen tablonun karşı tarafındaki tabloya (+) konarak yazılır. Dışına bağlantılar

tabloların herhangi birindeki kayıtların tümü listelenmek istendiğinde kurulur.

Örnek1 ;

```
SELECT ename, job, dept.deptno, dname
    FROM emp, dept
    WHERE emp.deptno (+) = dept.deptno;
```

```
ENAME      JOB       DEPTN      DNAME
---------- --------- ---------- --------------
CLARK      MANAGER           10 ACCOUNTING
KING       PRESIDENT         10 ACCOUNTING
MILLER     CLERK             10 ACCOUNTING
SMITH      CLERK             20 RESEARCH
ADAMS      CLERK             20 RESEARCH
FORD       ANALYST           20 RESEARCH
SCOTT      ANALYST           20 RESEARCH
JONES      MANAGER           20 RESEARCH
ALLEN      SALESMAN          30 SALES
BLAKE      MANAGER           30 SALES
MARTIN     SALESMAN          30 SALES
JAMES      CLERK             30 SALES
TURNER     SALESMAN          30 SALES
WARD       SALESMAN          30 SALES
                             40 OPERATIONS
```

Bu ornekte , "dept" tablosundaki tüm kayıtlar "emp" tablosuyla eslesme olsa da olmasa da listelenmistir.

Örnek2 ;

```
SELECT ename, job, dept.deptno, dname
    FROM emp, dept
    WHERE emp.deptno (+) = dept.deptno
        AND job (+) = 'CLERK';
```

```
ENAME      JOB       DEPTNO     DNAME
---------- --------- ---------- --------------
MILLER     CLERK             10 ACCOUNTING
SMITH      CLERK             20 RESEARCH
ADAMS      CLERK             20 RESEARCH
JAMES      CLERK             30 SALES
                             40 OPERATIONS
```

Burada , "emp" tablosundaki job alanında CLERK olanların listelenmistir. Ancak job alanında kullanılan (+) işareti ile ilk şartın yerine getirilmesi sağlanmıştır.

Dışına Bağlantı (Outer Join) kurulmasındaki bazı kurallar ;

- (+) işareti sadece WHERE cümleciğinde kullanılmaktadır.
- A ve B tablolarında çoklu bağlantı şartları kullanılmışsa, (+) işareti tüm koşul satırlarında olmak zorunda.
- (+) işareti ile IN koşulunda kullanılmaz, OR ile ayrılan şart cümleciğinde kullanılmaz,
- Karşılaştırma (+) işareti ile birlikte alt sorgulamalarda kullanılmaz.

Aşağıda verilmiş olan tablolar ve sorgulamalar konunun daha iyi anlaşılmasına yardımcı olacaktır.

```
SELECT custno, custname
    FROM customers;

CUSTNO     CUSTNAME
---------- --------------------
1          Angelic Co.
2          Believable Co.
3          Cabels R Us

SELECT orderno, custno,
    TO_CHAR(orderdate, 'MON-DD-YYYY') "ORDERDATE"
    FROM orders;

ORDERNO    CUSTNO     ORDERDATE
---------- ---------- -----------
      9001          1 OCT-13-1993
      9002          2 OCT-13-1993
      9003          1 OCT-20-1993
      9004          1 OCT-27-1993
      9005          2 OCT-31-1993

3 no.lu firmaya ait sipariş girilmemiştir.

SELECT orderno, lineno, partno, quantity
    FROM lineitems;

ORDERNO    LINENO     PARTNO     QUANTITY
---------- ---------- ---------- ----------
      9001          1        101         15
      9001          2        102         10
      9002          1        101         25
      9002          2        103         50
      9003          1        101         15
      9004          1        102         10
      9004          2        103         20

9005 no.lu siparişin alt detayları yoktur.

SELECT partno, partname
    FROM parts;

PARTNO PARTNAME
------ --------
   101 X-Ray Screen
   102 Yellow Bag
   103 Zoot Suit



SELECT custname, TO_CHAR(orderdate, 'MON-DD-YYYY') "ORDERDATE"
```

```
    FROM customers, orders
    WHERE customers.custno = orders.custno (+);

CUSTNAME             ORDERDATE
-------------------- --------------
Angelic Co.          OCT-13-1993
Angelic Co.          OCT-20-1993
Angelic Co.          OCT-27-1993
Believable Co.       OCT-13-1993
Believable Co.       OCT-31-1993
Cables R Us
```

Bu örnekte bütün müşterilere ait siparişlerin tarihleri listelenmesi istenmiştir. Dikkat edilirse "Cables R Us" isimli müşteriye ait sipariş olmamasına rağmen listede yer almıştır. (+) işareti "orders" table yanında olduğu için "customer" tablosundaki tüm kayıtlar listelenecektir.

```
SELECT custname,
TO_CHAR(orderdate, 'MON-DD-YYYY') "ORDERDATE",
partno, quantity
    FROM customers, orders, lineitems
    WHERE customers.custno = orders.custno (+)
    AND orders.orderno = lineitems.orderno (+);

CUSTNAME             ORDERDATE          PARTNO   QUANTITY
-------------------- -------------- ---------- ----------
Angelic Co.          OCT-13-1993           101         15
Angelic Co.          OCT-13-1993           102         10
Angelic Co.          OCT-20-1993           101         15
Angelic Co.          OCT-27-1993           102         10
Angelic Co.          OCT-27-1993           103         20
Believable Co.       OCT-13-1993           101         25
Believable Co.       OCT-13-1993           103         50
Believable Co.       OCT-31-1993
Cables R Us
```

Yukarıdaki örnek, bir önceki örneğin devamı olmakla beraber "orders" tablosunda yer alan bütün siparişleri listelemiştir.

```
SELECT custname, TO_CHAR(orderdate, 'MON-DD-YYYY') "ORDERDATE",
    quantity, partname
    FROM customers, orders, lineitems, parts
    WHERE customers.custno = orders.custno (+)
    AND orders.orderno = lineitems.orderno (+)
    AND lineitems.partno = parts.partno (+);

CUSTNAME             ORDERDATE        QUANTITY PARTNAME
-------------------- -------------- ---------- ------------
Angelic Co.          OCT-13-1993            15 X-Ray Screen
Angelic Co.          OCT-13-1993            10 Yellow Bag
Angelic Co.          OCT-20-1993            15 X-Ray Screen
Angelic Co.          OCT-27-1993            10 Yellow Bag
Angelic Co.          OCT-27-1993            20 Zoot Suit
Believable Co.       OCT-13-1993            25 X-Ray Screen
```

```
Believable Co.       OCT-13-1993           50 Zoot Suit
Believable Co.       OCT-31-1993
Cables R Us
```

Bu örnekte diğer örneklerin devamı olmakla beraber bütün parçaları da listelemiştir.

Bu örneklerin yorumlamasında "Believable" firmasına ait siparişin niye alt detayları yok ? "Cables" firmasına ait niye hiç sipariş yok ? Bunların sorularını cevaplayacak SQl cümlecikleri bu şekilde olacaktır.

==========

UNION **Bütün kayıtların seçilmesi**
UNION ALL **Bütün kayıtların seçilmesi (tekrar eden kayıtlarda dahildir)**
INTERSECT **İki sorgulama cümleciğinde de olan kayıtların seçilmesi**
MINUS **İlk sorgulamada olan ancak ikinci sorgulamada olmayan kayıtların seçilmesi**

==========

Yukarıdaki SQL fonksiyonları SELECT cümleciği ile birlikte kullanılmaktadır. Bu fonksiyonları kullanabilmek için sorgu cümleciklerinde aynı sayıda ki veri alanlarının aynı tipte olması gerekmektedir.

Sorguda kulllanılan veri tipi CHAR ise sonuç değer CHAR veri tipinde dönecektir.
Sorguda kulllanılan veri tipi VARCHAR2 ise sonuç değer VARCHAR2 veri tipinde dönecektir.

Aşağıdaki iki tablo konu içinde yer alan örneklerde kullanılacaktır.

SELECT part
FROM orders_list1;

PART
SPARKPLUG
FUEL PUMP
FUEL PUMP
TAILPIPE

SELECT part
FROM orders_list2;

PART
CRANKSHAFT
TAILPIPE
TAILPIPE

UNION

Aşağıdaki SQL cümleciğinde order_list1 tablosu ile order_list2 tablosu karşılaştırılarak aynı olan kayıtlar alınmamamıştır. Ayrıca ikinci tabloda olmayan alanın nasıl kullanıldığına dikkat ediniz.

```
SELECT part, partnum, to_date(null) date_in
    FROM orders_list1
UNION
SELECT part, to_null(null), date_in
    FROM orders_list2;

PART       PARTNUM DATE_IN
```

```
---------- ------- --------
SPARKPLUG  3323165
SPARKPLUG          10/24/98
FUEL PUMP  3323162
FUEL PUMP          12/24/99
TAILPIPE   1332999
TAILPIPE           01/01/01
CRANKSHAFT 9394991
CRANKSHAFT         09/12/02
```

Bu örnekte sadece "part" alanı kullanıldığından her iki tabloda yer alan aynı kayıtlar seçilmemiştir.

```
SELECT part
    FROM orders_list1
UNION
SELECT part
    FROM orders_list2;

PART
----------
SPARKPLUG
FUEL PUMP
TAILPIPE
CRANKSHAFT
```

UNION ALL

Bu fonksiyon, UNION aksine , aynı olan kayıtları da seçerek listeleyecektir.

```
SELECT part
    FROM orders_list1
UNION ALL
SELECT part
    FROM orders_list2;

PART
----------
SPARKPLUG
FUEL PUMP
FUEL PUMP
TAILPIPE
CRANKSHAFT
TAILPIPE
TAILPIPE
```

INTERSECT

İki sorgu cümleciğinde, ikisinde de yer alan kayıtları seçerek listeler.

Aşağıdaki örnekte, "orders_list1" tablosunda ve "orders_list2" tablosunda olan sadece TAILPIPE olduğu için tek kayıt listelenmiştir.

```
SELECT part
    FROM orders_list1
INTERSECT
SELECT part
    FROM orders_list2;

PART
----------
TAILPIPE
```

MINUS

İki sorgu cümleciğinden, ilkinde olup ta ikincide olmayan kayıtlar seçilerek listelenir.

Aşağıdaki örnekte, "orders_list1" tablosunda olupta "orders_list2" tablosunda olmayan "SPARKPLUG" ve "FUEL PUMP" olduğu için seçilerek listelenmiştir.

```
SELECT part
    FROM orders_list1
MINUS
SELECT part
    FROM orders_list2;

PART
----------
SPARKPLUG
FUEL PUMP
```

Bu komutlarla ilgili başka örneklerde verilmek istenirse ;

```
CREATE TABLE elemanlar (
      eleman_no   number(3),
      adi         varchar2(15),
      soyadi      varchar2(10)
);

CREATE TABLE mutseriler (
      Musteri_no  number(5),
      Unvan       varchar2(20),
eleman_no   number(3)
);
```

Şeklinde iki tablomuz olsun. “Elemanlar” tablosunda satış elemanları tanıtılmış, “Musteriler” tablosunda da eleman_no alanı ile bu musteriye atanmış satış elemanları tanıtılmıştır.

Hiçbir müşteriye ataması yapılmamış satış elemanlarının listesi nedir ?

Bu sorgunun cevabını verebilmek için “elemanlar” tablosunda olan ama “musteriler” tablosunda olmayan kayıtların listesini almak yeterlidir. Yani ;

```
SELECT eleman_no FROM elemanlar
MINUS
SELECT eleman_no FROM musteriler;
```

Müşterilere atanması yapılmış satış elemanlarının listesi nedir ?

Bu sorgunun cevabını verebilmek için “elemanlar” tablosu ile “musteriler” tablosunda aynı olan kayıtların listesini almak yeterlidir.

```
SELECT eleman_no FROM elemanlar
INTERSECT
SELECT eleman_no FROM musteriler;
```

TRIGGER

Veritabanında yer alan TABLO ve bu tabloya ait kayıtlar üzerinde INSERT, UPDATE ve DELETE komutlarının işlemesi öncesinde veya sonrasında devreye giren SQL komutlar dizine TRIGGER denir.

TRIGGER kelime anlamı olarak “Tetikleme” şeklindedir. Veritabanında yer alan tablolar üzerinde sadece kayıt ekleme, değiştirme ve silme işlemleri sırasında çalışacak olan SQL komutları topluluğudur.

INSERT, UPDATE ve DELETE komutu veritabanına gönderildiği zaman sunucu üzerinde yaratılmış olan trigger’lar ya işlemin öncesinde veya işlemin sonrasında çalışacaklardır.

COMMIT veya
ROLLBACK

AFTER
TRIGGER

BEFORE
TRIGGER

INSERT
UPDATE
DELETE

TRIGGER’ların kullanım amaçları ;

- Tablolar üzerindeki işlemlerin denetim ve kontrol altında tutulmasına yönelik,
- Tabloların alanlarına verilebilecek değerlerin kontrol edilmesi,
- Tablolar arasındaki ilişkilendirmelerin denetlenmesi,

şeklindedir. Trigger'lar sunucu üzerinde çalışacakları için yönetim ve değişiklik yapılması kolaydır. Böylece veritabanında yer alan tabloların alanlarında yer alacak değerlerin son bir kez denetlenmesi mümkün olmaktadır. Programlar tarafından kontrol edilmekle beraber özellikle çeşitli yerlerden tablolara işlem yapılması mümkün olduğundan kritik kontrollerin, denetlemelerin bu triggerlar üzerinde yapılması önerilmektedir.

Sunucuya gönderilen INSERT, UPDATE veya DELETE komutundan ÖNCE ve/veya SONRA trigger devreye girecektir. Bu esnada tablo üzerinde değişiklikler kontrol edilir ve COMMIT komutuna kadar bu işlemler diğer kullanıcılara yansıtılmaz.

Trigger içersinde yer alacak kontrollerin, denetimlerin kısacası işlemlerin her kayıt için yapılacağının bilinmesi lazımdır. Gereksiz hiçbir işlemin özellikle kontrollerin trigger içersinde yer almamasına dikkat edilmelidir. Çünkü her kayıt eklemede veya değiştirmede veya silmede trigger devreye girecek ve kontrolleri yapacaktır. Bu da sistemin performansını etkileyecektir. Genel olarak kullanıcıların işlem yapma zamanlarını etkileyeceğinden özellikle üzerinde çok INSERT, UPDATE ve DELETE işlemlerinin sıkça yapıldığı tablolarda etki komutları içeren trigger'lar kullanılmasına dikkat edilmelidir.

Trigger yaratma komutu aşağıdaki gibidir.

```
CREATE or REPLACE TRIGGER trigger_adi
 BEFORE INSERT OR DELETE OR UPDATE ON tablo_adi
 FOR EACH ROW
  BEGIN
    IF INSERTING THEN
        .............
    END IF;
    IF DELETING THEN
        .............
    END IF;
    IF UPDATING('alan_adi') THEN
        .............
    IF UPDATING THEN
        .............
    END IF;
  END;
```

Veritabanında herşey bir obje olduğundan ve bu objeler SYSTEM altında ilgili tablolar içerisinde tutulduğundan dolayı yaratılacak her trigger'a biri isim verilecektir. Trigger isimlendirilmesinde de anlaşılır, kolayca çözülebilir notasyonların kullanılması önerilmektedir. (Trg_biud_tablo_adi veya Trg_aiud_tablo_adi şeklinde...)

Burada CREATE komutuyla birlikte aynı zamanda REPLACE seçeneği de kullanılmıştır. Böylece ileride trigger üzerinde değişiklik yapılmak istendiği zaman var olan trigger değiştirilmiş olacaktır.

BEFORE INSERT or UPDATE or DELETE şeklindeki ifade ile bu trigger ; INSERT, UPDATE veya DELETE komutlarından ÖNCE çalışacaktır.

AFTER INSERT or UPDATE or DELETE şeklindeki ifade ile bu trigger ; INSERT, UPDATE veya DELETE komutlarından SONRA çalışacaktır.

ON tablo_adi ifadesi ile bu trigger'ın hangi tabloya ait olduğu belirtilmektedir.

FOR EACH ROW ifadesi ile bu trigger içindeki komutlar tabloda ki tüm kayıtlar için çalışacak anlamındadır.

IF INSERTING ifadesi ile bu trigger içinde sadece tabloya kayıt ekleme esnasında devreye girecek SQL ifadeleri yer alacaktır.

IF DELETING ifadesi ile bu trigger içinde sadece tablodan kayıt silinmesi esnasında devreye girecek SQL ifadeleri yer alacaktır.

IF UPDATING ifadesi ile bu trigger içinde sadece tabloda ki kayıtların değiştirilmesi esnasında devreye girecek SQL ifadeleri yer alacaktır.

IF UPDATING(‘alan_adi’) ifadesi ile bu trigger içinde sadece tablonun bir alanında değişiklik yapıldığında devreye girecek SQL ifadeleri yer alacaktır.

Görüldüğü üzere bir trigger içinde aynı anda tüm işlemlerin kontrolü ÖNCE’den (BEFORE) sağlanmış olacaktır. Aynı şekilde bu kontroller veya diğer tablolara etkileri SONRA’da (AFTER) olabilir.

Ø Trigger’lar içinde kullanılacak SQL komutlarının 60 satırı geçmemesi önerilmektedir.

Ø Tablo üzerindeki işlemlerin kontrolü sırasında tablo üzerinde başka bir işlemin (Insert, Update, Delete) devreye girmemesine dikkat edilmelidir. Bu takdirde aynı trigger devreye gireceğinden işlemler başarısız olur ve trigger hatası verilir.

Ø Trigger’ların kolayca yönetilmesi için çoğunlukla INSERT, UPDATE ve DELETE işlemleri için ayrı ayrı trigger’lar yazılması önerilir. Böylece bir tabloya ait en fazla 6 adet trigger olabilir.

Trigger komutu içinde kullanılan FOR EACH ROW ifadesi eğer kullanılmaz ise trigger içersinde tüm kayıtlar için değil, tablo genelinde kontroller yapılacak anlamındadır.

```
CREATE TRIGGER trg_biud_emp
    BEFORE DELETE OR INSERT OR UPDATE ON emp
    DECLARE
        kaysay INTEGER;
    BEGIN
        /* Cumartesi veya Pazar Kontrolü */
        IF (TO_CHAR(SYSDATE,'DY') = 'CMT' OR TO_CHAR(SYSDATE, 'DY') = 'PAZ')
         THEN raise_application_error( -20501,
       'Tatil günlerinde tablo üzerinde değişiklik yapılamaz');
        END IF;
       /* Resmi tatillerin kontrolü */
        SELECT COUNT(*)
         INTO dummy
          FROM tatil_gunleri WHERE gun = TRUNC(SYSDATE);
        IF dummy > 0
         THEN raise_application_error( -20501,
          'Resmi Tatil günlerinde tabloda değişiklik yapılamaz');
        END IF;
        /* 8:00 ve 18:00 kontrolü */
        IF (TO_CHAR(SYSDATE, 'HH24') < 8 OR  TO_CHAR(SYSDATE, 'HH24') >= 18)
          THEN raise_application_error( -20502,
       'Çalışma saatleri dışında tabloda işlem yapılamaz');
```

```
        END IF;
      END;
```

Yukarıda ki örnek trigger, EMP tablosu üzerinde INSERT, UPDATE ve DELETE işlemlerinden önce çalışacaktır. Ancak her kayıt için değil tablo genelinde devrededir.

Trigger içinde değişken kullanıldığından bu değişken adı ve tipi DECLARE satırında gösterilmiştir.

Eğer trigger içinde ki kontroller geçersiz sonuç verirse RAISE_APPLICATION_TRIGGER komutu ile hata mesajı kullanıcıya verilecek ve trigger'dan çıkılmış olacaktır. Diğer satırlar çalışmayacaktır.

Eğer bir trigger içinde SELECT ifadesi sonucunda bir değer ataması yapılacaksa bu SELECT .. INTO ifadesinin kullanılmasıyla olacaktır. Burada ;

```
SELECT COUNT(*)
  INTO kaysay
  FROM tatil_gunleri
 WHERE gun = TRUNC(SYSDATE);
```

DECLARE ifadesi ile tam sayı olarak tanımlanmış olan KAYSAY değişkenine, tatil_gunleri dosyasında ki GUN alanı ile işlem tarihi eşit olan kayıt sayıları COUNT(*) komutuyla bulunmuş ve INTO ifadesi ile değeri atanmıştır.

TRIGGER'lar veritabanında yaratıldıktan veya üzerinde değişiklikler yapıldıktan sonra derlenmeleri gerekmektedir. Eğer bir trigger içersinde yer alan bir tablo DROP edilmişse bu trigger geçersiz duruma düşecek ve o komut satırı çalıştığında hata vererek çalışması kesilecektir. Bunun için özellikle veritabanı yöneticilerinin sistem içersinde yer alan tabloları ve ilgili triggerları sürekli kontrol altında tutmaları gerekmektedir.

Trigger içersinde tabloda kullanılan alanların YENI (NEW) ve ESKI (OLD) değerleri aynı anda kullanılmaktadır.

Ø INSERT ifadesi sırasında alanların :NEW değeri bulunur. Tabloya yeni bir kayıt ilave edildiğinden alanların ilk değerleri yenidir ve :NEW olarak kullanılır. Alanın eski değeri null dur.
Ø DELETE ifadesi sırasında alanların :OLD değeri bulunur. Tabloda var olan bir kayıt silindiğinde alanların eski değerleri (:OLD) mevcuttur, yeni değeri null dur.
Ø UPDATE ifadesi sırasında alanların hem :NEW hem de :OLD değeri bulunur. Tabloda kayıtlarda üzerindeki alanlarda değişiklik yapıldığından alanın eski (İOLD) ve yeni (:NEW) değerleri mevcuttur.

Ancak alanların :NEW ve :OLD değerleri BEFORE TRIGGER işleminde geçerli olmaktadır çünkü AFTER TRIGGER'da tablodaki değişiklikler yapıldığından alanın yeni değerini kullanacaktır.

Trigger'larda BEFORE seçeneği daha çok kullanılmaktadır. Çünkü tabloya etki edecek değişikliklerin yapılmadan önce kontrol edilmesi gerek şarttır. AFTER seçeneği ise bir tablonun değişiminden sonra etkilemesi gereken başka bir tablolar varsa onların üzerinde işlem yapılmasını sağlayacaktır.

Bir trigger içersinde, veritabanında yaratılmış olan bir PROCEDURE veya FUNCTION çağrılması mümkündür. Özellikle tekrar eden durumlarda, SQL satır sayısının fazla olduğu durumlarda kullanılabilir.

Bir tablo üzerinde bir seferinde çok sayıda kayıt üzerinde işlem yapılmak istendiği

durumlarda veya girilmiş olan kayıtların tekrar düzenlemesi durumlarında, veritabanı yöneticisi trigger'ların çalışmasını geçici olarak durdurabilir. İşlemlerden sonra tekrar trigger'ı aktif edecektir. Bu işlemlerin, veritabanında kimsenin çalışmayacağı zamanda olması mutlak şarttır. Aksi takdirde, tabloların ve alanların bilgi girişleri güvenli olmayacak ve tutarsız değerler olabilecektir.

Trigger'ların çalışmasını geçici olarak durduran komut ALTER TRIGGER'dır.

ALTER TRIGGER Komutu

Trigger'ları devreden çıkaran veya devreye alan veya derleyeme amacıyla kullanılır.

```
ALTER TRIGGER [ENABLE] trigger_adi
              [DISABLE]
              [COMPILE]
```

ENABLE devre dışı (DISABLE) durumunda olan triggerı çalışmak üzere tekrar devreye alır
DISABLE trigger devre dışındadır, trigger içindeki komutlar çalışmayacaktır.
COMPILE trigger derlenecek ve varsa içersindeki hatalar kullanıcıya gösterilecektir.

```
CREATE TABLE musteriler (
      must_kod    varchar2(3),
      must_unvan  varchar2(15),
      must_tar    date,
      cre_user    varchar2(10),
      cre_date    date  DEFAULT sysdate,
      mod_user    varchar2(10),
      mod_date    date
);

CREATE TABLE islemler (
      islem_must  varchar2(3),
      islem_tar   date,
      islem_tut   number(15),
      cre_user    varchar2(10),
      cre_date    date  DEFAULT sysdate,
      mod_user    varchar2(10),
      mod_date    date,
) ;
```

Bu örnekte ISLEMLER tablosu (islem_must) ile MUSTERILER tablosu (must_kod) alanları ile bağlı olacağını düşünürsek ve bunun için FOREIGN KEY kullanılmadığı durumarda kontrollerin TRIGGER'lar içinde nasıl olduğunu görelim.

```
CREATE OR REPLACE TRIGGER trg_biud_musteriler
      BEFORE INSERT or UPDATE or DELETE ON musteriler
      FOR EACH ROW
      DECLARE
            Kaysay  INTEGER;
```

```
      BEGIN
            /* Musteri kaydinin silinmesinin kontrolu */
            IF DELETING THEN
                  SELECT count(*) INTO kaysay FROM islemler
WHERE islem_must = :old.must_kod;
IF kaysay>0 THEN
      RAISE_APPLICATION_ERROR('20501',
'Islemler dosyasinda kullanilan musteri silinemez');
                  END IF;
            END IF;
            /* must_kod alanin degisikliginin kontrolu */
            IF UPDATING('must_kod') THEN
RAISE_APPLICATION_ERROR('20501',
'bu alanda degisiklik yapilamaz');
            END IF;
            /* Kayit eklerken ilgili alanlara otomatik deger aktarilmasi */
            IF INSERTING THEN
                  :new.credate := SYSDATE;
                  :new.creuser := USER;
            END IF;
            /* Kayit degisikliginde ilgili alanlara otomatik deger aktarilmasi */
            IF UPDATING THEN
                  :new.moddate := SYSDATE;
                  :new.moduser := USER;
            END IF;
      END;
```

Dikkat edilecek noktalar ;

Ø MUSTERILER tablosundan kayıt silerken, ISLEMLER dosyasında kullanılıp kullanılmadığının kontrolü yapılmış ve en az bir kayıt bulunmuşsa kaydın silinmesi engellenmiştir.
Ø UPDATING('must_kod') satırı UPDATING satırından önce kullanılmıştır. Böylece alan değişikliğinin yapılmasına izin verilmeyecektir.
Ø Tabloya yeni kayit girisinde CREDATE ve CREUSER alanlarına sistem tarafından otomatik degerlerin atanmasi sağlanmıştır.
Ø UPDATING kontrolünde sadece MODUSER ve MODDATE alanlari güncellenmiştir. Kayıt yaratılma sırasında kullanılan CREDATE ve CREUSER alanlarında değişiklik yapılmamıştır.

```
CREATE OR REPLACE TRIGGER trg_biu_islemler
      BEFORE INSERT or UPDATE ON islemler
      FOR EACH ROW
      DECLARE
            Kaysay  INTEGER;
      BEGIN
            Kaysay := 0 ;
            /* Yeni giris kontrolu */
            IF INSERTING THEN
                  SELECT count(*) INTO kaysay FROM musteriler
WHERE must_kod = :new.islem_must;
IF kaysay=0 THEN
      RAISE_APPLICATION_ERROR('20501',
'Musteriler dosyasinda olmayan musteri eklenemez');
                  END IF;
```

```
                  :new.credate := SYSDATE;
                  :new.creuser := USER;
            END IF;
            /* must_kod alanin degisikliginin kontrolu */
            IF UPDATING('islem_must') THEN
RAISE_APPLICATION_ERROR('20501',
'bu alanda degisiklik yapilamaz');
            END IF;
            /* Kayit degisikliginde ilgili alanlara otomatik deger aktarilmasi */
            IF UPDATING THEN
                  :new.moddate := SYSDATE;
                  :new.moduser := USER;
            END IF;
      END;
```