YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

# ORACLE TABANLI VERİ YÖNETİMİ

**Mat.Müh. Pelin ŞENTÜRK**

**F.B.E Matematik Mühendisliği Anabilim Dalında Hazırlanan**

**YÜKSEK LİSANS TEZİ**



**Tez Savunma Tarihi** : 10.09.2001
**Tez Danışmanı** : Yrd.Doç.Dr. Ayla ŞAYLI
**Jüri Üyeleri** : Prof. Tahir ŞİŞMAN
: Prof.Dr. Mehmet AHLATÇIOĞLU

**İSTANBUL, 2001**

**İÇİNDEKİLER**

## KISALTMA LİSTESİ

| | |
|---|---|
| VTYS | Veritabanı Yönetim Sistemleri |
| DBMS | DataBase Management System |
| RDBMS | Relational DataBase Management System |
| SQL | Structured Query Language |
| PL/SQL | Procedural Language / Structured Query Language |
| DML | Data Manipulation Language |
| DDL | Data Defination Language |
| DCL | Data Control Language |
| SGA | System Global Area |
| PGA | Program Global Area |
| SNMP | Simple Network Management Protocol |

## ŞEKİL LİSTESİ

## ÇİZELGE LİSTESİ

## ÖNSÖZ

Lisans eğitimim sonrası çalışma hayatımla beraber yürüttüğüm yüksek lisans eğitimimde, hazırlamış olduğum tez konusunu seçebilmiş olmayı bir şans olarak nitelendiriyorum.

İş hayatımda bana referans niteliğinde olacak bu tezin kapsamı çok geniştir. Araştırma alanının bu kadar geniş olması gerek konu içeriklerinde ve gerekse alan tespitinde zorluklar içermiştir. Sonuç itibariyle Oracle tabanlı verinin yapısını ve yönetimini öğrenerek, stok modülünün oluşumundaki gerekli olan tüm sistem analizi ve kodlama aşamalarını gerçekleştirip amaçladığım noktaya varmaktan dolayı çok mutluyum.

Bu konuda çalışabilmemi sağlayan Yrd.Doç.Dr.Ayla Şaylı'ya yardımlarından ve desteğinden dolayı teşekkür ederim. Önerileriyle Ayla Hanım'la çalışma fırsatı veren Bölüm Başkanımız Prof.Tahir Şişman'a da teşekkürü bir borç bilirim.

Ayrıca desteklerini tüm öğrenim hayatımda hissettiğim anneme, babama ve kardeşim Tülin'e, anlayışlarından, bana sundukları imkanlarda dolayı Başak Hayat'a ve tezimin son aşamasına kadar beni destekleyen ve anlayışlarını esirgemeyen arkadaşımlarıma çok teşekkür ederim.

Pelin ŞENTÜRK

## ÖZET

Veritabanı teknolojisi bilgisayar bilimlerinde en hızlı gelişen alanlardan biridir. Bununla beraber bu araştırmada ilişkisel veritabanı sistemlerinin kavramları ve uygulamaları üzerine bir çalışma ortaya koymak amaçlanmıştır. İlişkisel bir veritabanı sistemi olan Oracle 8i ve uygulama geliştirmek için Developer 6i kullanılmıştır.

İlişkisel sistemin kavramları, fiziksel ve mantıksal veritabanı yapıları, veritabanı nesneleri, veritabanını yönetmek için kullanılan bellek ve süreç yapıları açıklanmıştır. İlişkisel veritabanı yaratımı, kontrolü ve yönetim sistemleri için kullanılan SQL (Structured Query Language-Yapısal Sorgulama Dili) tanımlanmış, veri üzerine yaptığı operasyonlar örneklerle aktarılmıştır. PL/SQL, Oracle'ın SQL'in bir uzantısı olan prosedürel dili, SQL*Plus aracı, veritabanında çalışacak uygulamaların geliştirilmesinde kullanılan Developer 6i , ASCII formatlı verinin veritabanına aktarımında kullanılan metodlar SQL* Loader yardımıyla anlatılmıştır.

Tez süresince Oracle tabanlı veri yönetimi esas alınarak stok takip modülü oluşumu amaç edinilmiştir. Yukarıda açıklanan tüm kavramlar ve araçlar bu modülün geliştirilmesi için kullanılmıştır. Ayrıca sistem analizi çalışmaları ilgili tüm kullanıcılardan alınan bilgiler ve istekler doğrultusunda gerçekleştirilmiştir. Genel olarak modül, stok giriş çıkışlarının yapılarak stoktaki malzemelerin asgari ve azami miktarları ışında stok hareketi, malzeme fiyatı, departman harcamaları ve talepleri takibini sağlar ve muhasebe ile entegre olabilecek bir yapıdır.

Tezin içeriğindeki stok modülü ve veritabanı yönetimi ile ilgili verilen tüm bilgiler örneklerle desteklenmiş ve birbirlerini referans edecek şekilde açıklanmıştır.



**Anahtar kelimeler:** İlişkisel veritabanı sistemleri, veritabanı yönetimi, veritabanı yapıları, veritabanı işlemleri, yapısal sorgulama dili, prosedürel dil, stok modülü,analiz, tasarım, kodlama.

**ABSTRACT**

Database tecnology is one of the most rapidly growing area of compute and information science. The aim of this research work is to give information to the concepts of a relational database system. Oracle 8i Enterprise Edtion for database system and Developer 6i for developing applications are used.

Concepts of relational database system, physical and logical database structures, database objects and memory and process structures were mentioned because of their necessity in managing relational database systems. The standart language for relational database management system,SQL (Structured Query Language), is described. And also SQL statements, that are used for all operations on the information in an database, are explained with examples. PL/SQL ,Oracle's procedurel language extention to SQL, SQL*Plus program, Developer 6i,a builder to develop applications, SQL*Loader, an utility to load data from ASCII fixed-format to database are explained.

Throughout the thesis, the intention has been the formation of stock follow-up module on the basis of oracle based data control. The terms and tools explained above have been all used to develop this module. Furthermore, the development of the module has been carried out in line with the data and requests received from end users about system analysis studies. The module, in general, provides follow-up of stock changes, material price, department expenses and requests, in light of the minimum and maximum quantities of materials in stock after stock ins and outs are completed. It is also suitable for an integration with accounting.

All the information that are used to mention the stock module and database management are described with examples supporting each other.

**Keywords:** Relational database systems, database management, database structures, database operations, structured query language, prosedurel language, stock module,analyze, design, codding.

# 1. GİRİŞ

Veritabanı yönetim sistemi (VTYS- Database Management System - DBMS) depolanan verilerin yönetimini ve bakımını sağlayan, birden çok sayıda kullanıcının eş zamanlı uygulama programlarıyla veriye erişip güncellemesini, değiştirmesini ve sorgulamasını sağlayan bir sistemdir.

## 1.1 Geleneksel Dosya-Tabanlı Veri İşlem Sistemleri

Geleneksel Dosya-Tabanlı Veri İşleme Sistemlerinde, bilgisayar programları belirli bilgi aktivitelerini desteklemek için dizayn edilirler. Veri dosyaları da bu programlara uygun formda verileri tutmak için oluşturulurlar. Veri dosyaları, onu kullanan programların verimli işlemesi için organize edilirler. Böyle tip sistemlerde veritabanı tek bir entegre yapı gibi bulunamaz ve birçok dosyaya birbirinden bağımsız işlem görür.

## 1.2 İlişkisel Veritabanı Yönetim Sistemileri

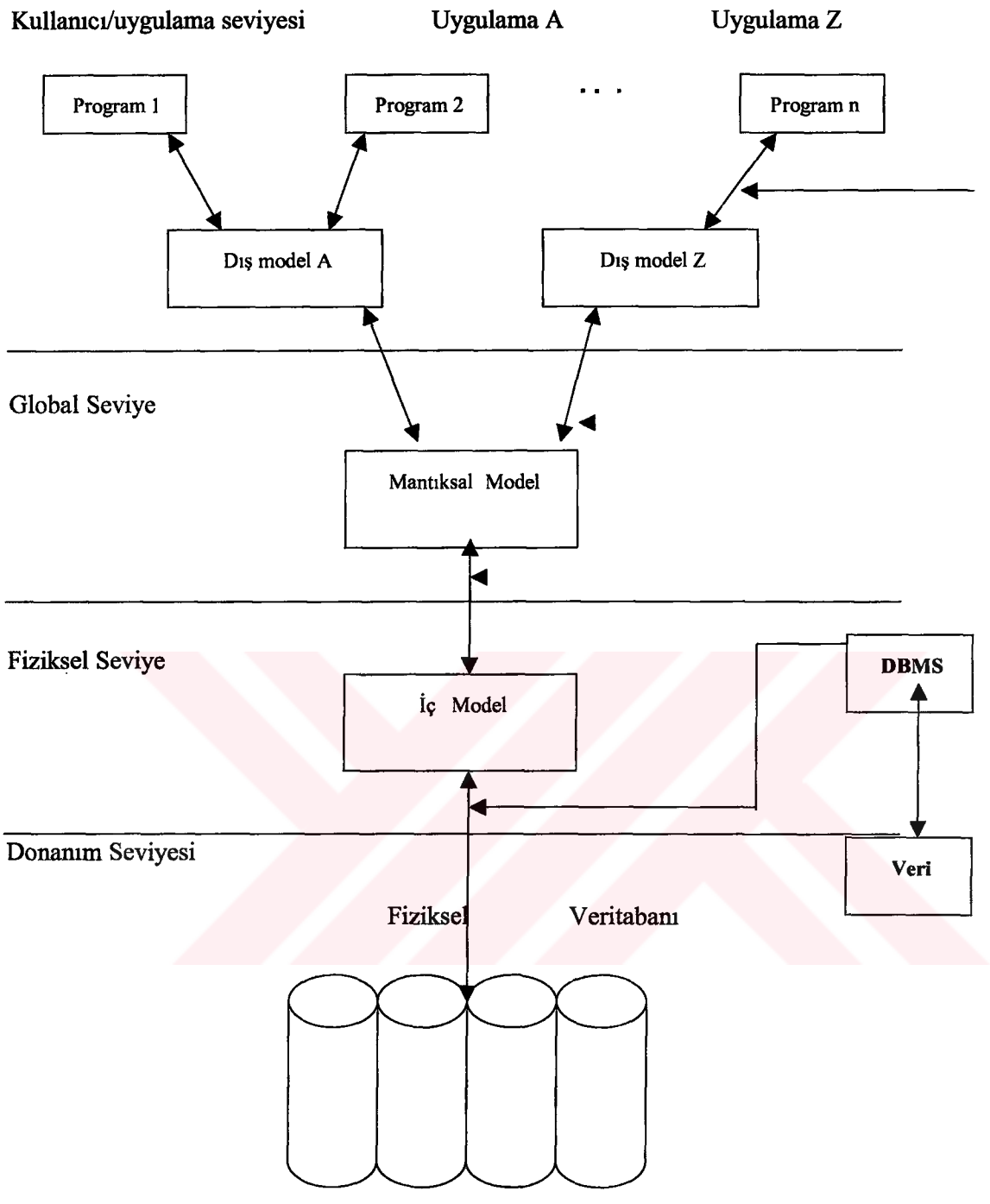
İlişkisel Veritabanı Yönetim Sistemi (Relational Database Management Systems – RDBMS) verilerin bütünlük içersinde güvenli bir şekilde tutulabildiği ve birden fazla kullanıcıya aynı anda verilere kolay ve hızlı erişim olanakları sağlayan sistemlerdir. Kompleks iş modellerinin ilişkisel veritabanında depolanması bu nesnel yapı sayesinde daha kolay gerçekleşmektedir. Oracle nesneye dayalı bir ilişkisel veritabanıdır.

Veritabanı yönetim sistemleri (VTYS) fiziksel, donanım, global ve kullanıcı/uygulama seviyelerinden oluşmaktadır (Şekil 1.1). Sistemin başarısı seviyelerin entegrasyonuna bağlıdır.

### 1.2.1 İlişkisel model

İlişkisel model 1970'lerin başlarında Edgar F.Codd tarafından formüle edilmiştir (Codd, 1970). İlişkisel model veritabanının teorik bir modelidir çünkü onunla veritabanını gösteren ve modelleyen bir dizi formül yaratılabilir. Bu özelliği ile veritabanı tasarımcıları ve araştırmacıları veritabanının özellikleri üzerinde bu yapıda çalışabilirler. Teorik olarak bu dizayndan veritabanı sistemini analiz ederek iyi bir sistem olup olmadığına karar verilebilir.

Bir veri modeli verilerin özelliklerini ve aralarındaki ilişkilerin gösterilmesini sağlar. Ayrıca veriye uygulanabilecek metotları gösterir. Bir ilişkisel model üç bölüm içerir:




Şekil 1.1 VTYS mimarisi

- Yapısal bölüm: Yapılar, veritabanındaki verileri depolayan ya da onlara ulaşımı sağlayan tanımlanmış nesnelerdir. Örneğin; tablolar, indeksler yapısal kısmın elemanlarındandır.

- Operasyon bölümü: Bir veritabanına yapılabilecek işlemlerin hepsi operasyon bölümünü luşturur. Veritabanından veri almak, güncellemek ya da veritabanının yapısında değişiklik yapmak operasyonun birer parçasıdır. Operasyonlar kullanıcıların veritabanının veri ve yapılarını yönetmeleri için tanımlanmış işlemleridir (Date, 1981). Ayrıca her bir operasyon önceden tanımlanmış bir takım kurallara bağlıdır.

- Bütünlük bölümü (integrity): Veritabanının uymak zorunda olduğu kurallar grubudur. Bütünlük kuralları, veritabanının veri ve yapılarına uygulanacak operasyonları yönetir ve işlemesini sağlayarak bütünlüğü korur.

Çizelge 1.1 İlişkisel veritabanı sistemlerinin karşılaştırması

| **Bileşenler** | **İlişkisel Yapı** | **Tablolar** | **Dosyalar** |
|---|---|---|---|
| Departman<br>Personel | İlişki | Tablo | Dosya |
| Muhasebe, Tahsilat, BİM,Personel_adi,<br>Maasi | Nitelik adı | Kolon Başlıkları | Alan adı |
| D1,D2,ALI,<br>200000000 | Nitelik Değeri | Kolon girişi | Veri başlığı |
| <ali,d1,200000><br><ahmet,d2,50000> | Tuple | Satır | Kayıt |

İlişkisel modelin yapısal kısmı veritabanının inşa edilmesi için şu yapıları kullanır;

1) Alan (domain)
2) Nitelik (attribute)
3) İlişki (relation)
4) Anahtar (key)

İlişkisel veritabanında her bir kolon bir alandan alınarak oluşur. Kolonlar birleşerek ilişkileri (tabloları) oluşturur. Bir ilişki tablo gibi alanlardan alınan bir grup değeri içeren yapıdır.

Nitelikler kolonlardır ve her birinin bir başlığı yani adı vardır. İlişkisel veritabanı birçok ilişkiden (tablodan) ,tablolar ise satırlardan oluşmaktadır.

Anahtarlar ilişkisel veritabanının temel bileşenidir. Veritabanındaki ilişkisellik yani bağlılık anahtarlar sayesinde oluşturulur. Bir anahtar da tablonun bir veya daha çok kolonu olabilir. Anahtar değerleri her bir satır için farklıdır. Anahtarın bu tanımı geleneksel dosyalara benzer ama tablolarda aynı anahtardan bir tane daha bulunamaz.

İlişkisel modelin operasyon kısmını veritabanına yapılabilecek işlemler oluşturur. Veritabanına uygulanabilecek operasyonlar operatörler yardımıyla da gerçekleştirilebilir. Veritabanından veri almak, veritabanının içeriğini değiştirmek ya da veritabanının yapısını değiştirmek gibi işlemler vardır. Bu bölüm matematiksel işlemleri ile ilişkisel cebrin bir parçası olarak ele alınıp incelenir ve analiz edilebilir (Codd, 1970).

Veritabanı için uyulması gereken kurallar bütünlüğü sağlanmaktadır. İki tip bütünlük kuralı vardır. Bunlar varlık bütünlüğü ve referans bütünlüğüdür.

1) Varlık bütünlüğü (entity integrity) kavramı her bir satırın tek olarak bir ismi vardır mantığıyla, her bir satırın kolonlarının bir ana anahtara (primary key) sahip olmasını öngörür. Ana anahtarlar null (boş) değerler içeremezler.

2) Referans bütünlüğü; tabloların bir yabancı anahtar (foreign key) ile diğer tablonun ana anahtarına bağlanmayı belirtir. Böylelikle tablolar arası bağ kurulmuş olur ve veriler kontrollü olarak kaydedilir.

### 1.2.2 Oracle'da veritabanı ve bilgi yönetimi

Oracle nesneye dayalı bir ilişkisel veritabanıdır. Nesnelerin üç bileşeni vardır;

1) İsim; nesnenin tipini tek olarak tanımlamayı sağlar.

2) Nitelik (attibute) ; Oracle veritabanından gelen veri tipleri ve kullanıcının tanımladığı veri tiplerini içerir. Nitelikler verilerin yapılarını modellerler.

3) Metot; PL/SQL de yazılmış veritabanında saklanan fonksiyon ve prosedürlerdir. Ayrıca C gibi bir dilde de yazılabilirler ancak veritabanı dışında saklanırlar. Her nesne tipinin veri tipinin belirteçlerine bağlı kalarak yeni bir nesne yapmaları için yapıcı metotları (constructer method) vardır.

Veritabanı sunucusu bilgi yönetimi problemlerinin çözümü için bir anahtardır. Genellikle, bir sunucu çok kullanıcılı bir ortamda büyük kapasiteli bilgileri güvenilir bir şekilde yönetmek zorundadır, çünkü birçok kullanıcı aynı veriye eş zamanlı olarak erişebilir. Tüm bunlar yüksek bir performans ile yürütülmelidir. Bir veritabanı sunucusu hem yetkilendirilmemiş erişimi önlemeli hem de başarısız iyileştirmeler için verimli çözümler üretmelidir.

#### 1.2.2.1 Oracle sunucusu

Sunucu bir veritabanından ve bir sunucu oturumundan (instance) oluşur. Oracle sunucusu aşağıdaki özelliklerle sahiptir:

**İstemci/sunucu ortam (ayrık işlem - distributed processing) :** Oracle, bilgisayar sistemini ve ağını daha avantajlı kullanmak için; veritabanı sunucusu ve istemci uygulama programları arasında işlemlerin bölünmesini sağlar. Veritabanı uygulamalarını işleten iş istasyonları verinin yorumlanması ve görüntülenmesine yoğunlaşırken, Veritabanı Yönetim Sistemini işleten bilgisayar ise tüm veritabanı sunucu sorumluluklarını yürütür.

**Büyük veritabanı ve alan yönetimi :** Terabytelarla ifade edilen çok büyük veritabanlarını da destekleyebilir.

**Çoklu kullanıcılı veritabanına eşzamanlı erişim :** Aynı veriye birçok kullanıcının eş zamanlı olarak farklı veritabanı uygulamaları ile erişimine olanak sağlar. Veri çekişmelerini minimize eder ve veri uyuşmasını garanti eder.

**Bağlanılabilirlilik :** Yazılımı farklı tipteki bilgisayarlara ve işletim sistemlerine karşı ağdaki bilginin paylaşılması için izin vermektedir.

**Yüksek erişim :** Sürekli çalışabilirliği sayesinde normal sistem operasyonlarını, örneğin veritabanı yedeklenmesi, veritabanı kullanımını bölmeden gerçekleştirir.

**Kontrollü erişim :** Veriye erişimini veritabanı veya alt-veritabanı seviyesinde kontrol altında tutabilir. Örneğin; yönetici belirli bir uygulamaya diğer işlemleri etkilemeden izin vermeyebilir.

**Açıklık, endüstri standartları :** Endüstride, veri erişim dili, işletim sistemi, kullanıcı ara yüzü ve ağ protokolleri için kabul edilmiş standartlara bağlıdır.

Ayrıca Oracle Basit Ağ Yönetim protokolünü (Simple Network Management Protocol - SNMP) sistem yönetimi için destekler. Bu protokol heterojen sistemleri tek bir yönetim ara yüzü ile idare etmeyi sağlar.

**Güvenliğin yönetimi :** Yetkilendirilmemiş veritabanı erişimleri ve kullanımını önlenir.

**Veritabanı bütünlüğü :** Oracle veri bütünlüğünü sağlar. Böylelikle kodlama ve yönetim kontrollerini azalır.

**Taşınılabilirlik :** Farklı işletim sistemleri ile çalışabilir. Oracle uygulama geliştiricileri az bir modifikasyonla ya da hiç gerek kalmadan taşınabilir.

**Uyumluluk :** Oracle yazılımları endüstri standartları ile uyumludur.

**Ayrık sistem :** Ağda, ayrık çevrelerde, Oracle fiziksel olarak farklı bilgisayarlarda olan veriyi bir mantıksal veritabanına kombine ederek ağdaki diğer kullanıcıların erişimini sağlar.

### 1.2.3 Oracle veritabanı ve yapısı

Veritabanı; belirli işlemden geçirilen bir grup verinin, depolanması ve ilgili bilgilerin alınması amacıyla oluşturulmuş bir ünitedir (Oracle, 1999a).

Oracle veritabanının bir fiziksel bir de mantıksal yapısı mevcuttur. Fiziksel ve mantıksal sunucu yapıları ayrıdır, fiziksel depolanan veri mantıksal depolama yapısına erişimi etkilemeden yönetilebilir.

Ayrıca Oracle veritabanı açık yani erişilebilir ya da kapalı yani erişilmez olabilir. Yani bir veritabanının bazı yönetimsel fonksiyonlar için kullanıcıların erişimi engellenebilir.

#### 1.2.3.1 Fiziksel veritabanı yapısı

İşletim sistemi dosyaları ile tanımlanan veritabanı yapısıdır. Her Oracle veritabanı şu üç tip dosyadan oluşur;

1) Veri dosyası (Datafile) : Bir veritabanında bir veya daha çok bulunabilir.
2) Yeniden yap kütük (Redo log) dosyası : Bir veritabanında iki veya daha çok tekrar bulunabilir.
3) Kontrol dosyası : Bir veritabanında bir veya daha çok kontrol dosyası bulunabilir.

Bu dosyalar gerçek anlamda veritabanı bilgisini fiziksel olarak depolar. Diğer bir değişle fiziksel bölüm veritabanının işletim sistemi tarafından görülen kısmıdır.

**Veri dosyası**

Oracle veritabanının en az bir veri dosyası vardır. Veri dosyaları veritabanının tüm verilerini

içerirler. Veritabanının mantıksal yapıları-tablo,indeks gibi veri dosyalarında fiziksel olarak depolanırlar.

Veri dosyalarının özellikleri;

- Bir veri dosyası yalnız bir veritabanına bağlıdır,
- Veritabanında boşluk kalmadığında veri dosyalarının otomatik olarak genişleyebilme özellikleri vardır,
- Bir veya daha çok veri dosyası bir araya gelerek tablo alanı (tablespace) denilen mantıksal üniteleri oluştur.

**Veri dosyası kullanımı**

Veritabanının normal operasyonları sırasında veri dosyasındaki veriler okunur ve veri dosyası Oracle'ın memory cache'inde saklanır. Örneğin bir kullanıcı veritabanındaki bir tablonun verilerine erişmek isterse ve istenilen bilgi veritabanının memory cache'inde yoksa, memory de saklanan uygun veri dosyasından okunur.

Değiştirilen veya yeni girilen verinin derhal veri dosyasına yazılması gerekmez. Diske erişim miktarını azaltmak ve performansı artırmak için veri bellekte havuzlanır ve hepsi bir kerede uygun veri dosyasına DBWn arka plan prosesi sayesinde yazılır.

**Yeniden yap kütüğü**

Her Oracle veritabanının en az iki tane yeniden yap kütüğü (redo log dosyası) vardır. Bu dosyaların başlıca görevi veriye yapılan tüm değişiklikleri kaydetmektir. Değişiklik yapılan veri veri dosyasına yazılır, bir hata durumunda yapılan modifikasyon redo log dosyalarından öğrenilebilir.

Veritabanını başarısızlıklara karşı korumada redo log dosyalarına çok kritik görevler düşer. Redo log dosyasının kendi içersinde bir hata ile karşılaşması olasılığı göz önüne alınarak, redo log dosyası farklı disklerde birden çok sayıda tutulabilir.

**Yeniden yap kütüğü kullanımı**

Redo dosyalardaki bilgiler, yalnızca sistem veya araç hatalarından kaynaklanan ve verilerin veri dosyalarına yazılmasına engel olacak durumlarda veritabanını iyileştirmek için kullanılır.

Örneğin; eğer beklenmeyen bir güç kesintisi nedeniyle veritabanı operasyonu birden sonlandıysa, hafızadaki veri veri dosyalarına yazılmamış olabilir. Bu durumda veri kaybı söz konusudur. Bununla beraber veritabanı tekrar açıldığı zaman kaybolan veriler redo loglar sayesinde korunur. Oracle redo dosyasının en son bilgilerini veri dosyalarına yansıtarak oluşacak hataları engeller.

**Kontrol dosyaları**

Her Oracle veritabanının bir kontrol dosyası vardır. Kontrol dosyaları veritabanının fiziksel yapısını betimleyen varlık bilgilerini içerirler. Örneğin şu bilgiler kontrol dosyasında bulunur;

- Veritabanının adı
- Veritabanının veri dosyaları ile redo dosyalarının yeri ve adları
- Veritabanının yaratılma tarihi

Redo dosyalarda olduğu gibi kontrol dosyalarının da koruma amaçlı olarak farklı disklerde kopyaları bulundurulabilir.

**Kontrol Dosyasının Kullanımı**

Oracle'da her zaman veritabanının bir instance'ı başlar, kontrol dosyası veritabanını ve veritabanı operasyonları işleyişinde açık olmak zorunda olan redo dosyaları tanımlamak için kullanılır. Örneğin veritabanında yeni veri dosyasının ya da redo dosyasının yaratılması gibi değişimlerde veritabanının kontrol dosyası otomatik olarak değişir. Ayrıca kontrol dosyaları veritabanının düzeltilmesinde de kullanılır.

### 1.2.3.2 Mantıksal veritabanı yapısı

Oracle veritabanının mantıksal yapısı şunlarla belirlenir;

- Bir veya daha çok tablo alanları (Tablespace)
- Veri blokları (Data blocks)
- Genişlemeler (Extents)
- Parçalar (Segments)
- Şema nesneleri ( Schema objects ) ; bu yapılar direk olarak veritabanındaki verileri işaret ederler.

- Tablo (Table)
- Görüntü (View)
- Sıra (Sequence)
- Kaydedilmiş Yordam (Stored procedure)
- Eşanlam (Synonym)
- İndeks (Index)
- Kullanıcı (User)
- Küme (Cluster)
- Veritabanı bağlantısı (Database link)

- Veri sözlüğü (Data dictionary)

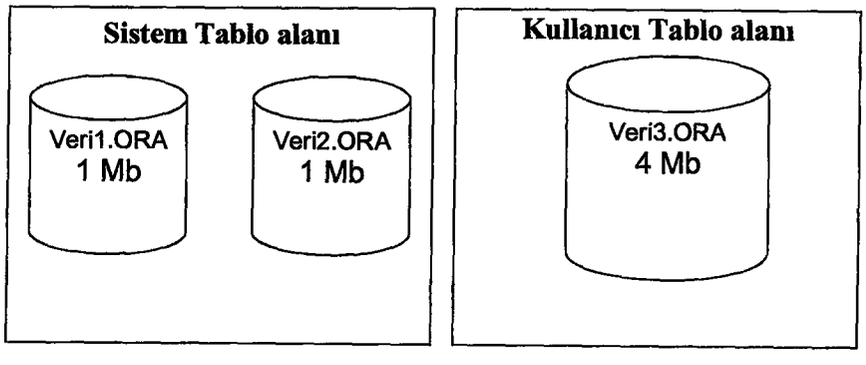
**Tablo alanı**

Bir veritabanı tablo alanı (tablespace) adı verilen mantıksal ünitelere bölünmüştür. Genelde yönetimsel uygulamaları sadeleştirmek için bir uygulamanın nesneleri grubu tablespace'lerdir. Her bir veritabanı bir veya daha çok tablespace'e bölünmüştür.

Veri dosyaları her bir veritabanı için bir veya daha fazla olabilir. Mantıksal yapı olan tablespace içindeki verileri fiziksel olarak depolar.

Tablespace'in veri dosyalarının toplam büyüklüğü tablespace'in toplam depolama kapasitesidir. Şekil 1.2'de sistem tablespace'i 2 MB yükleme kapasitesinde iken KULLANICI Tablespace'i 4 MB'dır.

Veritabanının tablespacelerinin toplam kapasitesinde veritabanının toplam depolama kapasitesini verir. Örnekteki veritabanının kapasitesi 6 MB'dır.




Şekil 1.2 Veritabanı, tablo alanı ve veri dosyası ilişkisi

**Veri Blokları, Genişlemeler ve Parçalar**

Oracle disk alanının kullanımını mantıksal depolama yapılarıyla kontrol altında tutar.

**Veri blokları**

Veritabanının verileri veri bloklarında depolanır. Bir veri bloğu diskte fiziksel veritabanının belirli bir sayıdaki byte'ına karşılık gelir. Blokların büyüklükleri veritabanı yaratılırken belirlenir. Bir veritabanı veri bloklarındaki boş veritabanı alanlarını kullanır. Veriler veri bloklarında tutulduğundan nesnelerin veritabanında kullandığı byte sayısı görülebildiği gibi blok sayısı da bilinir.

**Genişlemeler**

Mantıksal veritabanının bir sonraki seviyesi genişlemesidir. Bir genişleme veri bloklarının belirli ölçülerde devam eden kısımlarıdır. Bir veya daha fazla veri bloğundan oluşmaktadır. Belirli bir tip verinin depolanması için kullanılırlar.

**Parçalar**

Genişlemenin üstündeki mantıksal veritabanı depolama seviyesine segment adı verilir. Bir parça belli bir mantıksal yapı için kullanılan bir dizi genişlemedir. Her bir nesne için veritabanında bir segment ayrılır. Nesneye ilk ayrılan yer yetmezse, genişleme alarak büyür. Bir segment bir veya daha çok genişlemeden oluşmaktadır Segment tanımlanırken belirtilen maksimum genişleme sayısına kadar bu artış devam eder. Genişleme sayısı maksimum değeri aldığı zaman segment bir genişleme daha alıp büyüyemez.

- **Veri Segmentleri** Her bir non-clustered tablonun bir veri segmenti vardır. Tüm tabloların verileri veri segmentlerinin genişlemelerinde depolanır. Bölünmüş tablolarda (partitioned table) her bir bölümün bir veri segmenti vardır. Her bir cluster'ın bir veri segmenti vardır. Cluster'daki her tablonun verisi cluster'ın veri segmentinde depolanır.

- **Indeks Segmenti** Herbir indeksin verilerini depoladıkları bir indeks segmenti vardır. Bölünmüş indekslerin (partitioned index) herbir bölümünün bir indeks segmenti vardır.

- **Yapılandırma Parçaları(Rollback Segment)** "undo" bilgilerini depolamak için veritabanı yönetici tarafından yaratılan segment türüdür. Bir veritabanında bir veya daha

fazla rollback segment olabilir.

Rollback segmentteki bilgiler şu işlemlerde kullanılır;

- Veritabanı düzeltimi
- Kullanıcı için onaylanmamış işleri (uncommitted transactions) geri almak

- **Geçici Segment** Bu tip segmentler SQL cümleciklerin geçici iş alanına ihtiyaç duyduklarında icrayı (execution) tamamlamak için Oracle tarafından yaratılırlar. İcra tamamlandıktan sonra geçici segmentlerin genişlemeleri ilerde kullanılmak üzere sisteme döndürülürler.

Oracle devamlı olarak bulunan extentlerin segmentleri dolduğunda boşlukları ele geçerir. Bu yüzden extentlerin segmentleri dolduğunda, Oracle bu segment için başka bir extente ihtiyaç duyulduğu kadar el koyar. Çünkü segmentlerin extentleri diskte devamlı olmadıklarından, ihtiyaç duyulduğu kadar ele geçirilir.

**Şema nesneleri**

Şema veritabanı nesnelerinin bir grubudur. Şema nesneleri, direk olarak veritabanın verilerini işaret eden mantıksal yapılardan oluşur. Tablespace ile şema arasında hiçbir bağ yoktur. Şöyle ki aynı şemada bulunan nesneler farklı tablespacelerde yer alabilir ve bir tablespace farklı şemalardaki nesneleri içerebilir.

**Tablolar**

Tablo, Oracle veritabanında veri depolayan temel ünitelere verilen addır. Veritabanının tabloları kullanıcı erişimine olanak sağlar.

Tabloda veriler satır ve sütunlarda saklanırlar. Her tablonun bir tablo ismi vardır ve bir dizi kolondan oluşmaktadır. Her kolona bir isim, bir veri tipi ve genişlik (veri tipine göre verilmeyebilir, örneğin DATE tipindeyse genişliğe gerek kalmaz) verilir. Tablo yaratıldığında geçerli veri satırları kaydedilebilir, satırlar sorgulanabilir, silinebilir ve güncellenebilir.

Veritabanında bütünlük kurallarını sağlamak için tablolara bazı sınırlamalar ve triggerlar oluşturulabilir.

**Görüntüler**

Görüntüler; depolanmış sorgular olarak adlandırılabilir. View'ler gerçekten veri içermez ya da saklamazlar, tersine esas aldıkları tablolardaki (base tables) verileri kullanırlar. Base tablolar tablo yerine geçerler ve aynı zamanda base tablo bir view'dir.

Tablolar gibi, görüntüler de sorgulanabilir, güncellenebilir, kayıt girilebilir ve bazı kısıtlamalarla silinebilir ve sınırlandırılabilir.

Görüntüler şu işlemlerde kullanılır;

- Önceden belirlenmiş bir dizi kolon ve satıra erişimi kısıtlayarak, tablo güvenliğine ek bir seviye sağlar. Yani bazı kolonların görülmesini engelleyebilir. Örneğin; maaş bilgilerinin gizliğinden ötürü tablodaki maaş kolonunu görüntüde gösterilmez.

- Verinin karmaşıklığını gizlemek amacıyla da view'ler kullanılanır. Bir görüntü bir "join" yaratılmasında kullanılabilir. Böylelikle birbiriyle ilişkili fakat farklı tablolardaki kolon ve satırlar birleştirilebilir. Bununla beraber görüntü verinin birçok tablodan geldiğini gizler.

- Kullanıcılar için komutları sadeleştirir. Örneğin; görüntüler kullanıcıların bilgiyi çoklu tablolardan sorguların nasıl birleşip çalıştığını bilmelerine gerek kalmadan sorgulamalarını sağlar.

- Esas tablodan farklı bakışlardan veri gösterir. Örneğin; görüntüde esas alınan tabloları etkilemeden kolonları tekrar adlandırmak için view bir anlam sağlar.

- Karmaşık sorguları saklar. Böylelikle tablo bilgileri ile karmaşık işlemler yapan sorgular görüntü olarak saklanabilir ve görüntü sorgulandığı zaman hesaplamalar çalışır.

**Sıra**

Sıralar, veritabanının tablolarının sayısal kolonları için tekil olan bir seri sayısal liste üretir. Sıra otomatik olarak bir veya çoklu tablonun satırları için yaratılan tekil sayıları kullanarak uygulama programlarını sadeleştirir.

Örneğin; iki kullanıcı aynı zamanda bir tabloya yeni bir kayıt eklesin. Sıra kullanılarak her ikisine de tekil bir no verilebilir bu durumda kullanıcıların birbirlerini beklemesi söz konusu olmaz. Sequence her iki kullanıcı için doğru değeri üretir.

Sequence sayıları tablolardan bağımsızdır yani aynı numara bir veya daha çok tabloda kullanılabilir.

**Program ünitesi**

Gömülü prosedürler (stored procedures), fonksiyonlar, paketler, tetikleyiciler (triggers) ve anonim bloklar birer program ünitesidir.

Prosedürler ve fonksiyonlar bir dizi SQL ve PL/SQL'in belirli bir görevi icra etmek için bir araya geldikleri ifadelerdir. Ayrıca bu program üniteleri SQL'in kolaylığı ve esnekliğiyle yapısal programlama dilinin prosedürel fonksiyonu en iyi şekilde bileştirebilirler. PL/SQL kullanılarak, devamlı kullanım için prosedür ve fonksiyonlar veritabanında tanımlanabilir ve saklanabilir. Fonksiyonlar değer döndürme özelliklerinin dışında prosedürlerle aynıdır.

Paketler birbiriyle ilgili olan prosedürleri, fonksiyonları ve diğer paketleri birlikte bir ünite olarak veritabanında saklayan yapılardır. Paketlerin organizasyonel yararlar sağlarken ayrıca veritabanının performansını ve fonksiyonluğunu da artırır.

**Eşanlam**

Eşanlam (synonym), bir tablonun, görüntünün, sıranın ve program ünitesinin diğer adıdır. Bir synonym gerçekte bir şema nesnesi değildir ama direk olarak şema nesnesini işaret eder. Şu işlemlerde kullanılır;

- Bir şema nesnesinin gerçek adını ve sahibini gizlemek ,
- Bir şema nesnesine erişimi herkese açmak,
- Uzaktaki bir veritabanın tablosunu, görüntüsünü ve program ünitesini kullanılabilir kılmak,
- SQL ifadeleri veritabanı kullanıcıları için sadeleştirmek.

Bir synonym genel (public) veya özel (private) olabilir. Tek bir kullanıcı bir private synonym yaratarak bu eşanlamla yalnızca kendi erişimini mümkün kılabilir. Bir veritabanı yöneticisi ise genelde public synonym yaratarak esas şema nesnelerini sistem genelinde tüm kullanıcıların erişimini sağlar.

**İndeksler**

İndeksler tablolara bağlı olan seçimlik ve verilerin ele geçirilmesinde veritabanının performansını artıran yapılardır. Tablodaki veriler ulaşımı hızlandırmayı sağlarlar. Bir proses başlatıldığında Oracle istenilen satırlarla işlem yapılması için ulaşılabilen bazı ya da tüm indeksler kullanabilir. Tablodaki bir dizi veya belirli satır için sorgulama yapılacaksa indeksler bu gibi durumlarda çok yararlı olur.

İndeksler bir tabloda bir veya daha fazla kolon üzerinde yaratılabilir. Yaratılan indeks otomatik olarak Oracle tarafından beslenir ve kullanılır. Şöyle ki; tablodaki veri değişiklikleri ( yeni bir satır ekleme, satır güncelleme ya da silme) otomatik olarak alakalı tüm indeksleri yeniden düzenlenmesine neden olur.

İndeksler veriden hem mantıksal yönden hem de fiziksel yönden bağımsızdır. Böylelikle indeksler her zaman tabloyu ve diğer indeksleri etkilemeden silinebilir ya da yaratılabilirler.

**Kümeler**

Küme (cluster) tablodaki verinin elle geçirilmesinde performansı artırmak amacında seçimlik olarak yaratılan yapıdır.

**Kümesel Tablolar (Clustered Table)**

Kümeler ortak kolonları olan ve bunları beraber kullanıldıklarından dolayı fiziksel olarak beraber depolanan bir veya daha çok tablo grubudur. Beraber depolanmaları diske erişim zamanını azaltmaktadır.

Kümesel yapıda tabloların bağlantılı kolonlarına küme anahtarı (cluster key) denir. Küme anahtarı indekslenerek kümenin satırlarının minimum I/O içersinde elde edilmesini sağlar. Çünkü kümenin anahtarı çoklu tablolar için bir kere saklanır. Böylece küme bir grup tabloyu tek tek saklamaktan daha verimli olarak depolar.

İndeksler gibi kümeler de uygulama dizaynını etkilemez. Tablolar hem kullanıcılara hem de uygulamalara bir kümeye dahil olsalarda tekil bir tablo gibi görünürler. Kümesel tablonun içerdiği veriye ulaşmak kümesel olmayan tablonun verisine ulaşımdan farklı değildir.

**Veritabanı bağlantıları**

Veritabanı bağlantıları bir veritabanından diğerine geçiş yolunu tanımlayan bir şema nesnedir. Eğer bir global nesneye bir referans yapılırsa veritabanı bağlantıları kesinlikle kullanılır.

**Veri sözlüğü**

Her bir Oracle veritabanının bir veri sözlüğü vardır. Oracle veri sözlüğü bir dizi tablo ve görüntüyü içerirler ve yalnız okunan bu nesneler veritabanını işaret ederler. Örneğin bir veri sözlüğü veritabanının fiziksel ve mantıksal yapısı hakkında bilgi içerir. Ayrıca şu bilgileri depolar;

- Veritabanının geçerli kullanıcılarını,

- Veritabanında tablolar için tanımlanan bütünlüğü sağlayan kısıtlamalar hakkında bilgileri (integrty constraints)

- Şema nesneleri ne kadar alan işgal edildi ve bunların ne kadarı kullanımda olduğunu

Veri sözlüğü veritabanı yaratılırken yaratılır. Ayrıca veritabanının yapısında bir değişiklik yapıldığında veri sözlüğü otomatik olarak güncellenir.

#### 1.2.3.3 Veri işlemleri hizmetleri

Oracle; veritabanının alt kümesini bir veritabanından diğerine aktarmayı sağlayan üç hizmeti vardır.

- Veriyi veritabanından çıkarma Oracle da Export hizmeti sayesinde yapılmaktadır.

- Veriyi veritabanına sevketme işlemi Oracle 'da Import hizmeti sayesinde yapılmaktadır.

- Başka bir ortamda bulunan verilerin Oracle veritabanına yüklenmesi Oracle SQL*Loader hizmeti sayesinde yapılmaktadır.

**Veri çıkarma**

Veri çıkarma (Export) veri nesnelerinin Oracle veritabanları arasıda transferini sağlar. Özellikle yazılım ve donanım konfigürasyonlarının farklı olması transfer işlemini etkilemez. Export; nesnelerin tanımları ile tabloların verilerini Oracle veritabanından çıkarır ve onları ikili formatlı export dump dosyasında bir diskte ya da teypte saklar. Bu tür dosyalar ftp yoluyla kopyalanabilir, farklı bir ortama fiziksel olarak transfer edilerek kullanılabilir. Import ise ağa bağlı olmayan makinelerdeki veritabanlarına dump dosyasını transfer edilebilir. Export'un diğer işlevi de normal yedek alama prosedürüne ek olarak yedekleme yapabilmesidir. Bölüm 6.2'de bu konuya değinilmiştir.

**Veri alımı**

Veri alımının (Import) görevi Oracle veritabanından export hizmeti sayesinde çıkarılan veri nesnelerini farklı bir Oracle veritabanına kaydetmektir. Export dump dosyaları yalnızca Import tarafından okunabilir. Import; export ile çıkarılan veri tanımlarını ve tablo verilerini

içeren, diskte veya teypte bulunan ikili formatlı export dump dosyasını okur ve istenilen Oracle veritabanına kaydeder. Bölüm 6.3'de bu konu anlatılmıştır.

**SQL*Loader**

Export dump dosyaları yukarda da belirtildiği gibi yalnızca Import hizmeti sırasında okunabilir. Eğer ASCII formatlı bir dosyadan veri yüklenmeye ihtiyaç duyulursa, SQL*Loader kullanılır. SQL*Loader veritabanı dışındaki dosyalardan Oracle veritabanındaki tablolara veri yüklenmesinde kullanılır. Bölüm 6.1'de SQL*LOADER anlatılmıştır. SQL*Loader ile farklı formattaki giriş verileri alınabilir ve aynı yükleme oturumda birden çok tabloya yüklenebilir.

#### 1.2.3.4 Veri tipleri

Bir tablo veya görüntü yaratırken, bir fonksiyon veya prosedür oluştururken her bir argümanın bir veri tipinin olması gerekir. Veri tipleri tablonun kolonuna girilecek veriyi ve prosedürlerde kullanılacak parametreleri sınırlandırır. Veri tipleri şunlardır;

- CHAR
- DATE
- MLSLABEL
- NUMBER
- LONG
- LONG RAW
- RAW
- ROWID
- VARCHAR2

- **CHAR (n)** : Sabit uzunluktaki alfa nümerik (karakter) verilerinin tutulabildiği alan tipidir. Maksimum 255 karakter alınabilir. Kolon n ile belirtilen uzunluğundan daha kısa bir eğer alırsa Oracle kaydın sonuna boşluk ekleyerek sabit uzunluğuna getirir.

- **DATE :** Tarih tutulan alanlar için kullanılır. Ayrıca tarih bilgisine ek olarak saat bilgisi de tutulmaktadır. Tarih alanları ile aritmetik işlemler yapılabilir.

- **LONG :** 2 GB'a kadar karakter tutabilen alanlar için kullanılan veri tipidir. Bu uzunluk

işletim sistemine bağlı olarak 64KB'a kadar düşebilir. Bir tabloda birden fazla long veri tipili kolon bulunamaz. Ayrıca bu tip alanların üzerine indeks oluşturulamaz. Long alanlar şu SQL cümlelerinde kullanılamaz; where, group by, order by, distinct, create cluster, create table as select, substr, instr ve Sql cümlelerinin şart bölümlerinde.

- **LONG RAW :**2 GB' a kabar binary bilgi tutabilen alanlardır.

- **RAW :** Maksimum 255 byte'a kadar bilgi tutabilen binary alanlar için kullanılır.

- **ROWID :** Bir kaydın tekil adresini tutan alanlar için kullanılır. Veritabanında saklanan her kaydın bir adresi vardır. Bu değer SQL cümlesinde belirtilerek öğrenilebilir. Rowid üç bölümden oluşmaktadır. Bunlar; blok.kayıt ve dosya'dır.

- **VARCHAR2 (m) :** Değişken uzunluktaki karakter verileri tutan veri alanlarıdır. Maksimum 2000 karakter olabilir. Eğer m ile ifade edilen sayıdan daha kısa veriyi tutarsa Oracle char tipinin aksine verini sonuna boşluk eklemez. Böylece fazladan boşluk tutulmayarak yer kaybı önlenmiş olur.

## 2. YAPISAL SORGULAMA DİLİ

SEQUEL (Structured English Query Language – Yapısal İngilizce Sorgulama Dili), 1970 yılına Dr. E. F. Codd tarafından hazırlanan ilişkisel veritabanı yönetim sistemini (relational database managent system – RDBMS) kullanmak IBM tarafından için geliştirilmiştir. ( Codd, 1970) SEQUEL zamanla SQL olmuştur.1979 yılında şimdi ki adıyla Oracle o zaman ki Relational Software SQL'in ilk ticari uygulamasını geliştirmiştir. Artık günümüzde SQL bir RDBMS standardı olmuştur.

SQL aşağıdaki görevleri üslenmiştir;

- Veriyi sorgulama
- Tabloya yeni giriş, güncelleme ve satır silme
- Veritabanı nesnelerinin yaratılması, yer değiştirmesi, değiştirilmesi ve silinmesi
- Veritabanına ve nesnelerine erişim kontrolü
- Veritabanı tutarlılığını ve bütünlüğünü sağlama

SQL (Structured Query Language – Yapısal Sorgulama Dili) veritabanını tanımlayan ve idare eden bir programlama dilidir. SQL veritabanları ilişkisel veritabanlarıdır yani veriler bir dizi ilişki içersinde depolanır. Bir veritabanının en az bir tablosu (table) vardır. Her bir tablonun kolonları (column) ve satırları (row) vardır.

SQL'de üç tip komut vardır;

1) Veri İşleme Komutları (Data Manipulation Language – DML ) tablolardaki verilerin güncellenmesini, silinmesini ve ele geçirilmesini sağlar. En yaygın SQL DML komutu SELECT dir (Oracle, 1999d; s7-2).
   - SELECT
   - INSERT
   - UPDATE
   - DELETE
2) DDL : Veritabanı nesneleri üzerinde yapılan işlemlerde CREATE, ALTER ve DROP kullanılır. Bu işlemlerde kullanılan dile Veri Tanımlama Dili ( Data Defination Language – DDL ) adı verilir.
   - CREATE
   - ALTER

- DROP

3) DCL: Veritabanı kontrol dilinde (Data Control Language – DCL) yer alan komutlar sayesinde veritabanına erişim bir kontrol mekanizmasıyla denetlenir ve modifiye edilir. Kullanıcılar veritabanına bağlanırken ya da programlarla erişirken kullanıcı adı ve şifresiyle denetim altındalardır.

- GRANT
- REVOKE

## 2.1 SQL'in Temel Unsurları

### 2.1.1 Null

Eğer bir kolonun değeri yoksa kolon null'dır ya da null içermektedir denilebilir. Null tablolarda ana anahtar (primary key) ya da NOT NULL bütünlük kısıtlamasının olduğu kolonların dışında tüm kolonlarda olabilir (Oracle, 1999d; s2-49).

Sıfırın değeri olarak NULL gösterilemez, çünkü eşit değillerdir. Aritmetik ifadelerde null içeren kolon işlemin sonucunu da null yapar. Örneğin 100'e null eklenirse sonuç null olur. Diğer bütün operatörler de işlemin sonucunu null yaparlar. Yalnızca bağlama operatörü null dan etkilenmez.

Fonksiyonlara NVL ve TRANSLATE dışındakilere null argumanı verilince null döndürürler.

### 2.1.2 Sahte kolonlar

Sahte kolonlar tabloların kolonları gibi davranmakla beraber tabloda saklanmazlar. Bu kolonlar select edilebilir ama insert, update ve delete edilemez.

- CURRVAL ve NEXTVAL: Sıra şema nesnelerinin ürettiği tekil sıralı değerlerdir. Bu değerler genelde temel ya da unique anahtar için kullanılır. Sıra değerlerinden sahte kolonlar sayesinde yararlanılabilir.
  - CURRVAL sıranın geçerli değerini döndürür.
  - NEXTVAL sırayı artırarak bir sonraki değeri döndürür.

- Seviye : Hiyerarşi sorgularında kullanılır. Sahte kolon temel düğümde 1, çocukta 2, çocuğun çocuğunda 3 gibi değerler gönderir. START WITH ve CONNECT BY ile

kullanılır.

- ROWID Veritabanındaki her nesne için ROWID kolonu satırın adresini döndürür. Oracle satırı yerleştirmek için aşağıdaki bilgilere ihtiyaç duyar (Oracle, 1999d; s2-51).

  - Nesnenin numarası
  - Veri dosyasındaki hangi veri bloğunda yer aldığı
  - Veri bloğunun satır numarası (ilk satır 0 dır)
  - Veri dosyası (ilk dosya 1 dir)

ROWID'nin yararları;

- Bir satıra erişmek için en hızlı yoldur.
- Tablonun satırlarının nasıl depolandığını gösterir.
- Tablodaki satırların tekil temsilcileridir. Fakat ana anahtar olarak kullanılmamalıdır. Çünkü export ve import işlemlerinden sonra ROWID ler değişir. Ayrıca bir satırın silinmesiyle kullanılmayan ROWID yeni girilen satıra verilebilir.

- ROWNUM : Sorgulamada dönen satırlar için ROWID kolonu satırın tablodan seçim sırasına alır. İlk seçilen verinin ROWID değeri 1, ikincisinin 2 dir. Sorgulamada dönecek satırın sayısını kısıtlamada kullanılır. Örneğin;

```
SELECT * FROM envanter_table WHERE rownum < 50;

SELECT * FROM envanter_table WHERE rownum < 10 ORDER BY fiyat;
```

Yukarıdaki örnekte fiyatı en ucuz 10 envanteri geitirir.

## 2.2 Operatörler

### 2.2.1 Tekli ve ikili operatörler

Operatörler genel olarak iki grupta sınflanırlar.

1) Tekli operatör: Bir tekli operatörü yalnız bir operantı işler. Unary operatörü operantı ile aşağıdaki formatta kullanılır.

operatör operant

Parantez kullanarak operatörlerin öncelik sıralamaları değiştirilebilir. Oracle parantez içindeki işleme öncelik vererek parantez içinden dışına doğru ilerler.

Oracle ayrıca küme operatörlerini (UNION, UNION ALL, INTERSECT ve MINUS) destekler. Tüm bu operatörlerin öncelikleri birbirine eşittir.

Çizelge 2.1'de SQL operatörleri öncelik derecelerine göre azalan sırada listesi yer almaktadır.

Çizelge 2.1 SQL operatör öncelikleri

| Operatör | Operasyon |
|---|---|
| +, - | toplama, çıkarma |
| *, / | çarpma, bölme |
| +, - , \|\| | toplama, çıkarma, |
| =, !=, <, >, <=, >=, IS NULL, LIKE, BETWEEN, IN | karşılaştırma |
| NOT | değil |
| AND | birleşme |
| OR | ayrılma |

### 2.2.1.2 Aritmetik operatörler

Aritmetik operatörler nümerik değerlerin toplama, çıkarma, çarpma ve bölme işlemlerinde kullanılır. İşlemler sonucu elde edilen değer yine nümeriktir. Bu operatörlerden bazıları tarih

aritmetiğinde de kullanılır.

Çizelge 2.2 SQL aritmetik operatörleri

| Operatör | Amacı | Örnek |
|---|---|---|
| + - | Bunlar ikili operatörleridir. İfadenin negatif ya da pozitif olduğunu belirtir. | SELECT * FROM kod_table<br>WHERE isaret = -1 ; |
| * / | Bunlar ikili operatörlerdir. Çarpma, bölme. | UPDATE teklif_fiyat_tale<br>SET toplam_fiyat = fiyat * adet ; |
| + - | Toplama, çıkarma. Bunlar da ikili operatörlerdir. | SELECT toplam_fiyat– kdv_tutar FROM teklif_fiyat_table ; |

### 2.2.1.3 Birleştirme operatörü

Birleşme operatörleri karakter stringlere uygulanır.

Çizelge 2.3 SQL birleştirme operatörleri

| **Operatör** | **Amacı** | **Örnek** |
|---|---|---|
| \|\| | Karakter string birleşimi | SELECT ad1\|\|' '\|\|ad2<br>FROM sirket_table ; |

İki karakter string birleşmesiyle elde edilen sonuç da bir karakter stringdir. Eğer her iki karakter stringin tipi CHAR ise sonucun da veri tipi CHAR olur ve uzunluğu 2000 karakterle sınırlıdır. Eğer her ikisi VARCHAR2 ise sonuç da VARCHAR2 dir ve uzunluğu 4000 karakterle sınırlıdır.

Birleşme operatörü olan dikey çubuklar ( || ) farklı işletim sistemlerine geçişlerde sorun çıkarabilir. Bu nedenle Oracle CONCAT karakter fonksiyonunu geliştirmiştir. Dikey çubuklar gibi birleştirme işlemlerinde kullanılırlar ve farklı tipteki karakter kümelerine ya da farklı bir işletim sistemine geçişlerde sorun çıkarmazlar. Bölün sonundaki uygulamalar bölümünde örnek verilmiştir.

### 2.2.1.4 Karşılaştırma operatörleri

Karşılaştırma operatörleri bir ifadeyi diğeri ile karşılaştırır. Bu işlemin sonucu TRUE, FALSE veya bilinmeyendir. Bazı karşılaştırma operatörleri ;

=, <, <=, >, >=, !=

- BETWEEN ... AND...
- IN (liste)
- LIKE
- IS NULL
- IS NOT NULL
- NOT LIKE
- NOT IN
- NOT BETWEEN ...AND...

BETWEEN operatörü

Yüksek ve düşük iki sınır arasındaki değerlerin testi için kullanılır.

Örnek: Fiyatı 1.000.000 ile 5.000.000 arasındaki envanterler şu sorgu cümlesiyle bulunur;

```
SELECT * FROM envanter_table WHERE BETWEEN 1000000 AND 5000000 ;
```

**IN Operatörü**

Belirli bir listedeki değerlerin testi için kullanılır.

Örnek: Verilen tüm masa ve sandalye taleplerini getirmek için şu sorgu cümlesi kullanılır;

```
SELECT * FROM talep_table WHERE env_kod IN ('MS','SND') ;
```

NOT IN Operatörü

Belirli bir listedeki değerlerin dışındaki değerlerin getirilmesi için kullanılır.

Örnek: Verilen tüm masa ve sandalye taleplerinin dışındakileri getirmek isteyelim ama liste NULL değeri de ekleyelim.

```
SELECT * FROM talep_table WHERE env_kod  NOT IN ('MS','SND',NULL) ;
```

Bu sorgulama listedeki NULL sayesinde hiçbir zaman kayıt getirmez. Çünkü where şartı şu şekle dönüşür;

env_kod != 'MS' and env_kod != 'SND' and env_kod != null

LIKE Operatörü

Benzer değerlerin bulunmasında kullanılır. (%) karakteri bir veya birden fazla karaktere karşılık gelir. ( _ ) ise yalnızca bir karaktere karşılık gelmektedir. Eğer LIKE dan sonraki değerde bilinmeyen karakterlere karşılık gelen (%) ya da ( _ ) kullanılmamışsa eşitlik gibi davranır.

### 2.2.1.5 Mantıksal operatörler

- NOT Operatörü

Çizelge 2.4 NOT operatörünün durumları

| | **TRUE** | **FALSE** | **BİLİNMEYEN** |
|---|---|---|---|
| NOT | FALSE | TRUE | BİLİNMEYEN |

- AND Operatörü

Çizelge 2.5 AND operatörünün durumları

| AND | **TRUE** | **FALSE** | **BİLİNMEYEN** |
|---|---|---|---|
| TRUE | TRUE | TRUE | BİLİNMEYEN |
| FALSE | FALSE | FALSE | FALSE |
| BİLİNMEYEN | BİLİNMEYEN | FALSE | BİLİNMEYEN |

- OR Operatörü

Çizelge 2.6 OR operatörünün durumları

| AND | **TRUE** | **FALSE** | **BİLİNMEYEN** |
|---|---|---|---|
| TRUE | TRUE | TRUE | TRUE |
| FALSE | TRUE | FALSE | BİLİNMEYEN |
| BİLİNMEYEN | TRUE | BİLİNMEYEN | BİLİNMEYEN |

#### 2.2.1.6 Küme operatörleri

- UNION : İki ayrı SQL sorgusunun sonucunun birleşmesini sağlar.
- INTERSECT : İki ayrı SQL sorgusunun sonucunun kayıtlarının kesişim kümesini oluşturur.
- MINUS : Birinci sorgudan dönen kayıtlardan ikinci sorgudaki kayıtların fark kümesidir.

### 2.3 Fonksiyonlar

SQL cümlelerin kullanılabilen aldığı parametreler doğrultusunda bir değer döndüren ifadelere fonksiyonlar denir. Eğer bir fonksiyona gönderilen parametrelerde tip uyuşmazlığı varsa, Oracle bu değerleri fonksiyonda olması gereken tiplere çevirir (Oracle, 1999d; s4-1).

Eğer bir fonksiyon null argümanı ile çağrılırsa, CONCAT, DECODE, DUMP, NVL, REPLACE fonksiyonları dışındaki fonksiyonlar null döndürür.

Fonksiyonlar çalışma şekillerine göre ikiye ayrılabilir.

1) Tek Kayıt Fonksiyonları
2) Grup Fonksiyonları

#### 2.3.1 Tek kayıt fonksiyonları

Tek satır üzerinde işlem yapan fonksiyonlardır. Dört tip vardır;

1) Nümerik Fonksiyonlar
2) Karakter Fonksiyonları
3) Tarih Fonksiyonları
4) Dönüşüm Fonksiyonları

#### 2.3.1.1 Nümerik fonksiyonlar

Nümerik fonksiyonlar diğer adıyla sayı fonksiyonları number parametreler alır ve geriye nümerik sonuç gönderirler. Bazı sayı fonksiyonları;

- ABS(n) :Girilen "n" değerinin pozitif karşılığını sonuç olarak getirir.

- ACOS(n) : Girilen "n" değerinin arkkosinus karşılığını sonuç olarak getirir.

- ASIN(n): Girilen "n" değerinin arksinus karşılığını sonuç olarak getirir.
- ATAN: Girilen "n" değerinin arktanjat değerinin sonuç olarak getirir.
- AVG(n) :Girilen "n"'in ortalama değerini sonuç olarak getirir.
- CEIL(n): Girilen "n" değerinden büyük en küçük tam sayı değerini sonuç olarak getirir.
- COS (n) : Girilen değerin kosinus karşılığını sonuç olarak getirir
- COSH(n) : Girilen değerin hiperbolik kosinüs karşılığını sonuç olarak getirir.
- EXP(n) : e'nin n.üssünü sonuç olarak gönderir.
- FLOOR(n): n sayısından küçük veya eşit en büyük tam sayıyı döndürür.
- LN(n): n sayısının logaritmasını döndürür.
- LOG(m,n) : n sayısının m tabanına göre logaritmasının sonuç olarak gönderir.
- MOD(m,n) : "m" sayısını "n" ile böler ve kalanı sonuç olarak getirir.
- POWER(m,n) : "m" değerinin "n"inci kuuvvetini sonuç olarak getirir.
- SIGN(n) : Eğer n<0 ise (-1) değerini sonuç olarak getirir. Eğer n=0 ise(0) değerini, n>0 ise (1) değerini sonuç olarak getirir.
- SIN(n) : Verilen değerin sinüs karşılığını sonuç olarak getirir.
- SINH(n) : Hiperbolik sinüs karşılığını sonuç olarak getirir.
- SQRT(n) : Karekökünü sonuç olarak getirir.
- TAN(n) : N değerinin tanjant karşılığını sonuç olarak getirir.

- TANH(n) : Girilen değerin hiperbolik tanjant karşılığını sonuç olarak getirir.

### 2.3.1.2 Karakter fonksiyonlar

- CHR(n,[USING NCHAR2_CS]) : Farklı bir veritabanında farklı bir national karakter setindeki n değerinin ikilik karşılığı sonuç olarak getirir.

- CONCAT(char1,char2) : Verilen iki char tipini birleşmiş olarak getirir.

- INITCAP : Char tipi verinin ilk harfini büyük, diğerlerini küçük tipte sonuç olarak döndürür.

- INSTR : Char tipi birinci dizide, verilen char2 tipindeki verinin m'inci defa görüldüğü andaki, char tipi birinci veride kaçıncı eleman olduğunu gösterir.

- INSTRB : INST fonksiyonu ile aynıdır, fakat "n" ve sonuç olarak verilen değer bytes karakterindedir

- LOWER : Verilen char tipi veriyi bütün karakterleri küçük olarak getirir.

- LTRIM : Char tipi verinin verilen diziye benzer sol tarafındaki karakterleri benzer olmayandiziye rastlayıncaya kadar kaldırır.

- NLS_INITCAP : Char tipi verinin ilk harflerini büyük, diğerlerini küçük olarak bırakarak verir.

- NLS_LOWER : Verilen char tipi veriyi bütün karakterleri küçük olarak geri getirir.

- NLS_UPPER : Bütün karakterleri büyük olarak sonuç getirir.

- REPLACEVerilen char karakterinde, girilen her dizini değiştirilmesi istenen dizi ile değiştirip, sonuç olarak getirir.

- RPAD : Verilen char1 değerini, char2 dizinini kullanarak istenen karakter uzunluğuna çıkarır.

- SUBSTR : Char verisinin m'inci karakterinden başlayarak n adet karakterini sonuç olarak verir.

- SUBSTRB : SUBSTR ile aynı işlevi görmektedir, tek farkı m ve n değerlerinin bytes ile belirlenmesidir.

- TRANSLATE :Char dizinin içindeki her değerin, verilen karşılığının bulunduğu satırı sonuç olarak getirir.

#### 2.3.1.3 Tarih fonksiyonları

- ADD_MONTH(m,n) : m tarihine n ay eklenmesi ile oluşan tarihi sonuç olarak döndürür.

- LAST_DAY(tarih1) : tarih1 tarihindeki en son günü sonuç olarak döndürür.

- MONTHS_BETWEEN(tarih1,tarih2) : İki tarih arasındaki ay sayısını döndürür.

- NEXT_DAY(tarih1,karakter) : karakter denilen değişken haftanın günüdür ya da kaçıncı günü olduğu bilgisidir.

- ROUND(tarih1) : Günü bazında yuvarlama yapar.

- ROUND(tarih1,'MONTH') : Ay bazında yuvarlama yapar. Ayın 15'inden küçük bir günse aynı ayın başına yuvarlar. Ayın 15'inden büyük bir günse bir sonraki aynı ayın başına yuvarlar.

- ROUND(tarih1,'YEAR') : Yıl bazında yuvarlama yapar. Altıncı aydan küçükse aynı yılın başına yuvarlar. Altıncı aydan büyükse bir sonraki yılın başına yuvarlar.

- TRUNC(tarih1) : Günü bazında kesme yapar.

- TRUNC(tarih1,'MONTH') : Ay bazında kesme yapar. Ayın ilk gününe indirir.

- TRUNC(tarih1,'YEAR') : Yıl bazında kesme yapar. Yılın ilk gününe indirir.

#### 2.3.1.4 Dönüşüm fonksiyonları

- CHARTOROWID(char) : CHAR veya VARCHAR2 veri tipindeki bir değerin, ROWID veri tipindeki karşılığını sonuç olarak getirir.

- CONVERT(char,dest_char_set,[,source_char_set] : Bir karakter dizinini bir karakterden diğerine dönüştürür. Char dönüştürülecek değerdir. dest_char_set ise çevrileceği karakter kümesidir. source_char_set dönüştürülecek verinin bulunduğu veritabanının karakter setidir.

- HEXTORAW : Hexadecimal digid barındıran char tipi veriyi ham değere çevirir.

- RAWTOHEX : Girilen değeri hexadecimal eşdeğerini içeren bir karakter değere dönüştürür.

- ROWIDTOCAHR : Rowid değeri VARCHAR2 datatipindeki değere dönüştürür. Bu dönüşümün sonucu her zaman 18 karakterlidir.

- TO_CHAR :Sayı tipindeki veriyi, istenilen formata çevirmek için kullanılır.

- TO_LOB : LONG veye LONG RAW değerleri LOB değerlere çevirir.

- TO_NUMBER : Char tipi veriyi sayısal değere çevirir.

- TRANSLATE...USING : Tanımlanmış dönüşüm eşleniklerini kullanarak verilen text dökümünü dönüştürür.

### 2.3.2 Grup fonksiyonları

Grup fonksiyonlar birden fazla kayıt üzerinde işlem yaparak değer döndüren fonksiyon tipidir.

- AVG(kolon) :Kolon değerlerinin ortalama değerini sonuç olarak getirir.

- COUNT([ *| [DISTINCT | ALL]] exp) : Sorgudaki satır sayısını sonuç olarak getirir.

- MAX (kolon) : Kolon değerlerinin maksimum karşılığını sonuç olarak getirir.

- MIN(kolon) : Kolon değerlerinin minimum karşılığını sonuç olarak getirir.

- SUM(kolon) : Kolon değerlerinin toplamını sonuç olarak getirir.

## 2.4 Veri İşleme Dili

Veritabanındaki kayıtların üzerinde ekleme, silme, güncelleme ve sorgulama gibi işlemler yapmayı sağlayan dile veri işleme dili ( data manipulation language – DML ) denir. Bazı DML komutu şunlardır;

- SELECT
- INSERT
- UPDATE
- DELETE

### 2.4.1 Select komutu

Select bir veya daha çok tablo ya da görüntüden belirtilen kolonların verilen koşullara uygun olarak alınmasında kullanılan bir SQL komutudur. SELECT kayıtlar üzerinde değişiklik yapmaz, sadece veritabanındaki kayıtları sorgular. SQL'de en sık kullanılan komutlardan birisidir.

Komutları yapıları anlatılırken kullanılan sembollerin anlamı;

[] Kullanılması zorunlu olmayan ifadeleri belirtmektedir. {} Kullanılması zorunlu olan bölümü gösterir. "…" ise önceki bölümün istenildiği kadar tekrar edilebileceğini ifade eder. | Aynı anda sağındaki ve solundaki ifadelerden sadece birinin kullanılabileceğini gösterir. … Virgülden önceki bölümün istenildiği kadar tekrar edilebileceğini gösterir.

SELECT komutu yapısı;

```
SELECT [ALL | DISTINCT ] { * | [<nesne_ismi>].<kolon_ismi>,…}
FROM { <nesne_ismi> [ @<bağlantı_ismi>] [<diğer_isim>],…}
[WHERE <koşul_bölümü> [<altsorgu>]]
[GROUP BY { <kolon_ismi>,…}]
```

[HAVING <koşul_bölümü>]

[ORDER BY { <sütun sırası> | <diğer_ismi> | <kolon_ismi>,...} [ASC | DESC]

- SELECT : Listelenecek alanların belirtildiği bölümdür.
- FROM : Listelenecek alanların seçildiği tablonun belirtildiği bölümdür.
- WHERE : Sorgu şartlarının yazıldığı bölümdür.
- ALL : Tablodaki bütün kolonların listeleneceğini ifade eder.
- DISTINCT: Yenilenen kayıtların sadece birer tane listelenmesini sağlar.
- * from dan sonar yer alan tablo ve görüntülerin tüm alanlarının listeneceğini belirtir.
- nesne_ismi : Tablo ve görüntülerin ismidir.
- kolon_ismi: Tablo ve görüntülerin kolon isimleridir.
- bağlantı_ismi: Veritabanının bağlantısını belirtir.
- diğer_isim: Sütunlara verilen ikinci isimdir.
- koşul_bölümü: Sorgunun koşuludur.
- kolon_sırası: select bölümünde ilgili sütunun kaçıncı sıra olduğunu gösterir.
- Altsorgu : şart bölümünde yer alan SQL cümlesidir.
- GROUP BY : Sorgu sonucu dönen kayıtları gruplama işleminin yapıldığı bölümdür.
- HAVING : Grup halinde gelen SQL'lerde grubu ilgilendiren şart cümlelerinin yazıldığı bölümdür.
- ORDER BY : Sorgu sonucu dönen kayıtların sıralamasının yapılmasını sağlar.
- ASC : Kayıtların küçükten büyüğe doğru sıralanmasını belirtir.
- DESC : Kayıtların büyükten küçüğe doğru sıralanmasını belirtir.

### 2.4.2 Insert komutu

Tablo ve görüntülere kayıt girilmesini sağlayan veri işleme komutudur.

Yapısı şöyledir;

INSERT INTO { <nesne ismi> @ <baglanti ismi>} [<kolon ismi>,...]

VALUES (<deger>,...)

VALUES : Tabloya girilecek verilerin belirtildiği bölümdür.

INSERT komutu bir tabloya başka tablonun satırlarını kopyalayabilir. Yapı şu şekilde değişir;

INSERT INTO { <nesne ismi> @ <baglanti ismi>} [<kolon ismi>,...]

```
SELECT seçilen_kolonlar
FROM tablolar;
```

Burada INTO dan sonra belirtilen kolon sayısı ve tipi SELECT sonrası seçilen_kolonlar ile bir bir denk olmalıdır.

### 2.4.3 Update komutu

Tablo ve görüntüdeki alanları değiştirmede kullanılan veri işleme komutudur.

Yapısı;

```
UPDATE { <nesne_ismi> @ <baglanti_ismi>} [<diger_ismi>]
SET (<kolon ismi>,...) = { <kolon_ismi>,...| <altsorgu>}
WHERE <koşul_bölümü>
```

- SET : Değişecek alanları yazıldığı bölümdür.
- WHERE : Şartları yazıldı bölümdür.

### 2.4.4 Delete komutu

Tablo ve görüntülerdeki kayıtları slimeyi sağlar. Kullanım şekli şöyledir;

```
DELETE FROM { <nesne_ismi> @ <baglanti_ismi>} [<diger ismi>]
WHERE <koşul_bölümü>
```

## 2.5 Veri Tanımlama Dili

Veritabanı nesnelerini, yaratma, değiştirme ve silme işlemlerini, hak ve rollerin verilme ve alınma işlemlerini ve tablo, indeks ve cluster'ların analiz işlemlerini yapan komutların oluşturduğu dile Veri Tanımlama Dili ( DDL ) denir. Bütün bu komutlar veritabanının yapısını etkilediğinden dolayı veri sözlüğünü de değiştirmektedirler.

Sekiz tane ana DDL komutu vardır.

- CREATE
- ALTER
- DROP
- GRANT
- INVOKE
- ANALYZE

- AUDIT
- COMMENT

### 2.5.1 Create komutu

Nesne yaratmak için kullanılan bir DDL komutudur.

#### 2.5.1.1 Tablo yaratımı

Create Table komutu tablo yaratmak için kullanılır. Kullanım şekli şöyledir;

```
CREATE TABLE          <tablo ismi>
( <kolon ismi>        veri_tipi    [<kolon kısıtlaması],
  <kolon ismi>        veri_tipi    [<kolon kısıtlaması],
  ...                                                  )
[ <tablo kısıtlaması>]
[ PCTFREE             sayı ]
[ PCTUSED             sayı ]
[ INITRANS            sayı ]
[ MAXTRANS            sayı ]
[ TABLESPACE          <tablespace ismi> ]
[ STORAGE             kayıt parametreleri ]
[ RECOVERABLE | UNRECOVERABLE ]
[ AS                  <altsorgu> ]
```

Ek 1'de CREATE TABLE örnekleri verilmiştir.

#### 2.5.1.2 Tablo değişimi

Tanımlı tabloların değiştirilmesi DDL komutları sayesinde yapılmaktadır. Alter Table komutunun kullanımı için kullanıcının ALTER ANY TABLE hakkına sahip olması gerekmektedir. Alter table ile kısıtlamaların erişimi durdurulabilir ya da tekrar aktif olmaları sağlanabilir. Tekil ve ana anahtarlar oluşturulurken sistem otomatik olarak tablo üzerine indeks oluşturur. Tablodaki tekikler enable ya da disable edilebilir. Yeni yabancı anahtarlar eklenebilir. Kolon değiştirilebilir, silinebilir. Ancak boş (null) olan kolonlara not null kısıtlaması verilemez. Ayrıca kolonlar büyütülebilir ama küçülemezler.

Yapısı;

ALTER TABLE <tablo_ismi>

ADD | MODIFY | DROP (<kolon_ismi> <kısıtlama>)

Ek 1'de ALTER TABLE örnekleri verilmiştir.

#### 2.5.1.3 Tablo silme

Drop Table, tabloların veritabanından silinmesini sağlayan bir DDL komutudur. DROP ANY TABLE hakkına sahip olan kullanıcılar tarafından kullanılabilir. Yapısı;

DROP TABLE <tablo_ismi> [CASCADE CONSTRAINTS]

CASCADE CONSTRAINTS sayesinde tablolar arası referans kısıtlamaları sayesinde silinen tabloyla bağlantılı tüm tablolar silinir.

Ek 1'de DROP TABLE örnekleri verilmiştir.

#### 2.5.1.4 Eşanlam yaratımı

Eşanlam (synonym) tablonun, görüntünün, sıranın, prosedürün, paketin alternatif ismine verilen addır. Eşanlamlar sayesinde şemalara bağlı olan nesnelerin bağımsızlığı sağlanır ve farklı bir veritabanında olmalarının birbirlerine erişim konusundaki kısıtlamalarını ortadan kaldırır. Public anahtar kelimesi sayesinde bir nesne tüm kullanıcıların erişimine açılabilir. Ek 1' de CREATE SYNONYM örnekleri verilmiştir.

Yapısı;

CREATE [PUBLIC] SYNONYM <şema_ismi.eşanlam_ismi>

FOR şema_ismi.nesne_ismi@veritabanı_adı

#### 2.5.1.5 Eşanlam silme

Ek 1'de DROP SYNONYM örnekleri verilmiştir. Yapısı şu şekildedir;

DROP [PUBLIC] SYNONYM <şema_ismi.eşanlam_ismi>

#### 2.5.1.6 Kullanıcı yaratma

Create user komutuna Bölüm 3'de 3.3 Veritabanı kullanıcıları kısmında yer verilmiştir.

#### 2.5.1.7 Rol yaratma

Create role komutuna Bölüm 3'de 3.4.3.1'de yer verilmiştir.

#### 2.5.1.8 Rol değişimi

Create role komutuna Bölüm 3'de 3.4.3.2'de yer verilmiştir.

#### 2.5.1.9 Hak verme

Grant komutuna Bölüm 3'de 3.4.1 ve 3.4.2'de yer verilmiştir.

#### 2.5.1.10 Hakkı geri alma

Revoke komutuna Bölüm 3'de 3.4.1 ve 3.4.2'de yer verilmiştir.

### 2.6 Uygulamalar

1) Depatman kodlarını ve personel bilgilerini tutan iki tablo yaratalım.

```
CREATE TABLE depart_table
( depno number(3),
  depart_adi varchar2(25))

CREATE TABLE personel_table
( perno number(3),
  personel_adi varchar2(25),
  personel_soyadi varchar2(25),
  depno number(3),
  maas number,
  bagli_perno number(3))

CREATE TABLE personel_kimlik_table
(perno number(3),
 kimlik_tipi varchar2(3),
 kimlik_no number(10))

CREATE TABLE kimlik_tip_table
(kimlik_tipi varchar2(3),
 aciklama varchar2(25))
```

Personel_Table : Personelin departmanı, yöneticisi ve maaşı bilgilerini içerir.

Depart_Table : Departman kodlarını içerir.

Personel_Kimlik_Table : Personelin kimlik bilgilerini saklar.

Kimlik_Tip_Table : Kimlik tiplerinin tanımlandığı tablodur.

2) DESCRIBE komutu ile tablonun kolonlarını listeleyebiliriz.

```
DESCRIBE DEPART_TABLE
Name                      Null?    Type
------------------------- -------- ----------------------------
depat_no                  not null number(3)
depart_adi                not null varchar2(25)
```

3) Bir tablodaki tüm bilgilerin şartsız listelenmesi.

```
SELECT *
FROM DEPART_TABLE;
```

1 BİLGİ İŞLEM

2 TEKNİK

3 ÜRETİM

4 TAZMİNAT

5 MUHASEBE

6 TAHSİLAT

7 İNSAN KAYNAKLARI

8 İÇ HİZMETLER

4) SQL'de aritmetik işlemlerin uygulanması ve kolonlara yeni isim verme.

```
SELECT PERNO, PERSONEL_ADI ADI ,PERSONEL_SOYADI SOYADI, MAAS*12
YILLIK_MAASI
FROM PERSONEL_TABLE;
```

| PERNO | ADI | SOYADI | YILLIK MAAŞ |
|---|---|---|---|
| 5 | ALİ | GENÇ | 50000000000 |
| 2 | TAMER | ÖZTÜRK | 60000000000 |
| 1 | OYA | GERÇEK | 12500000000 |
| 3 | SEDA | BİRLİK | 8000000000 |

5) Sorgulamada kolonların birleştirilmesi.

"||" işareti ile kolonlar birleştirilir.

```
SELECT PERNO, PERSONEL_ADI||''||PERSONEL_SOYADI
FROM PERSONEL_TABLE;
```

| PERNO | PERSONEL_ADI\|\|''\|\|PERSONEL_SOYADI |
|---|---|
| 5 | ALİ GENÇ |
| 2 | TAMER ÖZTÜRK |
| 1 | OYA GERÇEK |
| 3 | SEDA BİRLİK |

6) Kayıtların tekrarlanmadan listelenmesi.

```
SELECT DISTINCT DEPTNO
FROM PERSONEL_TABLE;
```

| DEPTNO |
|---|
| 1 |
| 2 |
| 3 |
| 4 |

7) Sorgu sonucu dönen satırları sıralama

```
SELECT PERNO, PERSONEL_ADI ,PERSONEL_SOYADI , MAAS*12
```

```
FROM PERSONEL_TABLE
ORDER BY MAAS;
```

| PERNO | PERSONEL_ADI | PERSONEL_SOYADI | MAAS*12 |
|---|---|---|---|
| 3 | SEDA | BİRLİK | 8000000000 |
| 1 | OYA | GERÇEK | 12500000000 |
| 5 | ALİ | GENÇ | 50000000000 |
| 2 | TAMER | ÖZTÜRK | 60000000000 |

8) SQL'de sorgu şartlarını yazma ve operatörler

SQL'de şartlar where bölümünde mantıksal operatörler, SQL operatörleri ve küme operatörleri kulanarak belirtilir.

Yıllık maaşı 13.000.000.000 TL den büyük ve muhasebe departmanında çalışan personelleri bullamak için şu SQL cümlesi yazılır;

```
SELECT PERNO, PERSONEL_ADI ,PERSONEL_SOYADI , MAAS*12 YILLIK_MAAS
   FROM PERSONEL_TABLE
     WHERE YILLIK_MAAS>13000000000
      AND DEPTNO = 1;
```

| PERNO | PERSONEL_ADI | PERSONEL_SOYADI | YILLIK_MAAS |
|---|---|---|---|
| 5 | ALİ | GENÇ | 50000000000 |
| 2 | TAMER | ÖZTÜRK | 60000000000 |

9) Tabloların birleştirilmesi. (JOIN)

Personelin çalıştıkları departmanları ile listelenmesi.

```
SELECT A.PERNO, A.PERSONEL_ADI , A.PERSONEL_SOYADI , A.DEPTNO ,B.ACIKLAMA
FROM PERSONEL_TABLE A, DEPART_TABLE B
WHERE A.DEPT_NO = B.DEPT_NO;
```

| A.PERNO | A.PERSONEL_ADI | A.PERSONEL_SOYADI | A.DEPTNO | B.ACIKLAMA |
|---|---|---|---|---|
| 5 | ALİ | GENÇ | 1 | BILGI İŞLEM |
| 2 | TAMER | ÖZTÜRK | 1 | BILGI İşLEM |
| 3 | SEDA | BİRLİK | 2 | TEKNİK |
| 1 | OYA | GERÇEK | 3 | ÜRETİM |

# 3. ORACLE KAVRAMLARI

## 3.1 Oracle Mimarisi Bileşenleri

Oracle Sunucu bilgi yönetimine açık, kapsamlı ve entegre bir açı sağlayan ilişkisel veritabanı yönetim sistemi nesnesidir (Oracle, 1999a).

### 3.1.1 Temel bileşenler

Bir Oracle Sunucu 'da birçok proses, bellek yapısı ve dosyalar vardır (Oracle, 1999b). Bununla beraber bunlar hepsi bir SQL statement prosesi için kullanılmaz. Bazıları da veritabanının performansının artırmak için, yazılım ya da donanım sorunlarından veritabanını korumak için ya da veritabanının bakımı için gerekli diğer görevlerinde kullanılır. Bir Oracle Sunucu bir Oracle instance ve bir Oracle veritabanından oluşur.

### 3.1.2 Oracle oturumu

Oracle otrumu (instance) bir grup arka plan prosesleri ve bellek yapılarından oluşur. Bir instance veritabanındaki veriye erişim ile başlar. Bir instance başlatıldığında Sistem Global Alanı (System Global Area - SGA) ele geçirilir ve Oracle arka plan prosesleri başlatılır.

SGA , bellek alanıdır. Veritabanı prosesleri tarafından paylaşılan veritabanı bilgilerini saklar. Veri ve Oracle sunucu için kontrol bilgilerini içerir. Oracle Sunucusu olan makinenin virtual belleğini kullanır (Oracle, 1999b; s7-2). Şekil 3.1'de bir instance , prosesler ve diğer veritabanı dosyaları arasındaki ilişki gösterilmiştir.

SGA şu veri yapılarından oluşur:

- Veritabanı Tamponu: En son kullanılan veriler bu tampon bölgesine saklanır. Bunlar veri dosyalarına yazılan ya da bu dosyalardan okunan verilerdir.

- Redo Log Tamponu : Sunucunun ve arka plan proseslerinin veritabanına yaptığı değişiklikler burada izlenir.

- Paylaşım Havuzu (Shared Pool): En son execute edilen SQL statementler, veri sözlüğünden kullanılan bilgiler bu bölümde saklanır.

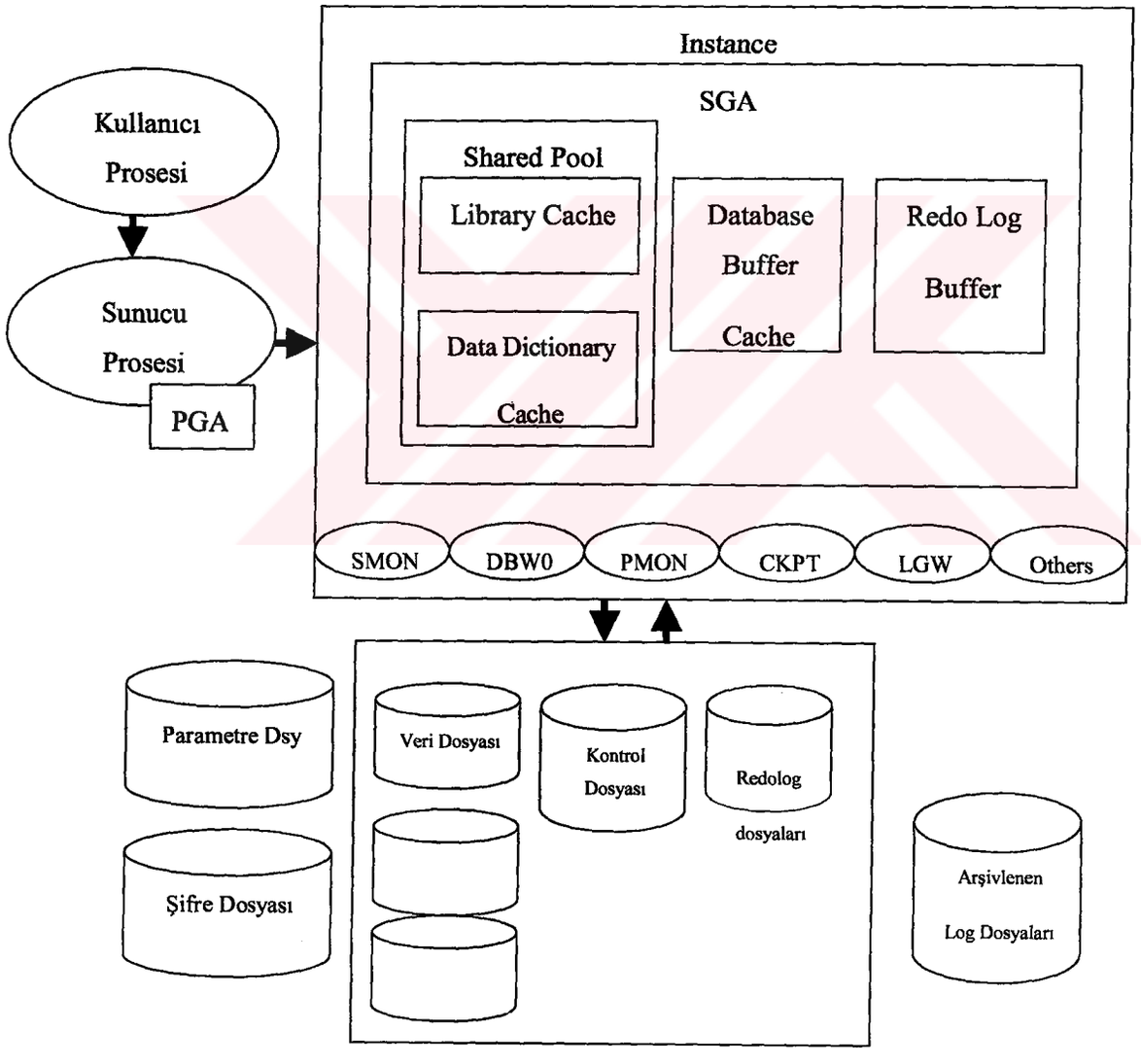
    - Kütüphane cache (Library cache)

- Veri sözlüğü cache (Data dictionary cache)

SGA da ayrıca iki tane seçimlik veri yapısı vardır:

- Java Havuzu (Java Pool): Java kodların saklanmasında kullanılır.

- Büyük Havuz (Large Pool): Direk SQL statementlarla ilişkili değildir, yedekleme ya da yükleme işlemlerinde veri bloklarının kopyalanmasında kullanılır.




Şekil 3.1 Oracle bellek ve proses yapısı

### 3.1.2.1 Oracle'da proses tipleri

**Kullanıcı işlemleri**

Uygulama programının icra etmesi için kullanıcı işlemi (proses) yaratılır ve idare edilir. Sunucu prosesle iletişimi kullanıcı prosesler yönetir. Kullanıcı prosesler sunucu proseslerle ile ara yüz programı sayesinde irtibat kurar (Oracle, 1999b; s8-2).

**Arka plan işlemleri**

Arka plan prosesleri ayrıca fonksiyonları işlerler.Her bir kullanıcı için ayrı ayrı çalıştırılan fonksiyonlarını birleştirir. Arka plan prosesleri O/I işlemler yaparlar ve daha iyi bir performans için diğer Oracle proseslerin paralelliğini artırır. Her bir instance için bir dizi arka plan prosesi yaratılır. Bu prosesler;

DBWn, LGWR, CKPT, SMON, PMON, ARCn, RECO, SNPn, QMNn dir.

**Sunucu işlemi**

İstemci/Sunucu sistemlerde kullanıcı ve sunucu prosesleri ayrı makinalardaexecute edilebilirler.

**SQL ifadesinin işlemi**

SQL ifadesinin tipine bağlı olarak Oracle sunucu bileşenleri kullanılır.

- Sorgular satır getirirler.
- DML ifadeleri logları değiştirirler.
- Onaylama (commit) işlemin (transaction) gerçekleşmesini sağlar.

Bazı Oracle sunucu bileşenleri SQL ifadesinin işlenmesinde görev almaz. Kullanıcı ve sunucu prosesleri Oracle Instance'a bir kullanıcının bağlanmasında kullanılır. Bu işlemler Oracle instance'ın bir parçası değildir ama SQL ifadesinin işlenmesinde ihtiyaç duyulur.

Bazı arka plana prosesleri, SGA yapıları ve veritabanı dosyaları SQL ifadesinin işlenmesinde kullanılır. Bu ifadenin tipine bağlı olarak farklı bileşenler kullanılır.

- Sorgulamalar kullanıcıya veri getirmek için ek proseslere ihtiyaç duyarlar.
- Veri İşleme Dilinde (Data Manipulation Langulage – DML) veriye yapılan değişikliklerin loglara kaydı için ek işlemlere gerek vardır.

- Commit işlemi de değişime uğramış verinin onaylanmasını sağlar.

Bazı arka plan prosesleri de direk olarak SQL ifadesinin prosesinde yer almaz ama veritabanını performansının artırılmasında kullanılır. Ayrıca seçimli bir arka plan prosesi olan ARC0 veritabanının iyileştirilmesini sağlar.

**Veritabanına bağlanma**

Bir instance'a bağlanmak için kullanılan prosesler:

Kullanıcılar bir SQL ifadesini Oracle'a yollamadan önce bir instance'a bağlanmak zorundadır. Kullanıcılar SQL*Plus aracı ile ya da uygulama geliştirme aracı olan Oracle Form ile kullanıcı prosesinde işlemler gerçekleştirirler.

En basit ayarlarda, bir kullanıcı Oracle sunucusuna bağlanınca Oracle Sunucusu tarafında bir proses yaratılır. Bu proses bir Sunucu Prosesdir. Sunucu proses Oracle instance ile istemci tarafında çalışan kullanıcı prosesin iletişimini sağlar ve kullanıcını yerine SQL ifadesini icra eder.

**Bağlantı**

Bir bağlantı, kullanıcı proses ile Oracle sunucu arasındaki iletişim yoludur. Bir veritabanı kullanıcısı sunucusuna üç yolla bağlanabilir.

1) Kullanıcı Oracle Instance'ın çalıştığı işletim sistemine bağlanır ve bu sistemde veritabanına ulaşmak için bir uygulamayı ya da aracı kullanır.

2) Kullanıcı yerel bilgisayarında bir uygulamayı ya da aracı başlatarak, diğer bir ağda çalışan Oracle instance'a bağlanır. Bu tip bağlantıda İstemci-Sunucu tipinde bir ağ yazılımı sayesinde kullanıcılar Oracle Sunucu 'a bağlanmış olur.

3) Üç Katmanlı Bağlantı: Bu bağlantı tipinde kullanıcı, Oracle instance'ın çalıştığı makineye ağ üzerindeki bir uygulama (application) ya da ağ (network) sunucusu sayesinde bağlanır. Örneğin, bir ağ üzerindeki kullanıcı browser çalıştırarak Nt sunucu'daki uygulamayı kullanır ve bu makinenin üzerinden Unix üzerindeki Oracle veritabanından verileri alır.

**Oturum**

Oturum (session), bir kullanıcının Oracle Sunucusuna olan belirli bir bağlantısına denir. Bir oturum kullanıcının sunucusu tarafından geçerli kılınmasıyla, bağlantının kurulmasıyla başlar

ve kendi isteğiyle bağlantıyı bitirmesiyle ya da beklenmeyen sonlardırmalarla bitirilebilir. Kullanıcı araçlarla, uygulamalarla birçok terminal üzerinden eşzamanlı oturumlarla kurabilir.

Burada açıklanan bağlantı tipleri sunucu ve kullanıcı proses arasında bire bir yürütülürler ve bu bağlantılara dedicated server connection denir. Eğer multithreaded sunucu (MTS) kullanılsaydı, birçok kullanıcı prosesi sunucu proseslerini paylaşabilirdi.

**Sorgulama işlemi**

Sorgular diğer tip SQL ifadelerinden farklıdır çünkü diğerleri sadece başarılı ya da başarısız olarak geri bildirimde bulunurlarken eğer sorgu hatasız ise sonuç olarak veri getirecektir.

Bir sorgunun işlenmesi üç aşamadan oluşur.

1) Parse
2) Execute
3) Fetch
4) Parse : Bu safhada SQL ifadesi kullanıcı parsesinden sunucu prosese geçer ve paylaşılmış SQL bölgesine (Shared SQL Area) yüklenir.

Parse esnasında sunucu proses şu işlemleri yapar.

- SQL ifadesinin bir kopyasını paylaşım havuzunda (shared pool) arar.
- SQL ifadesinin yazılımını kontrol ederek, ifadeyi geçerli kılar.
- Veri sözlüğüne tablo ve kolon tanımlamalarını onaylamak için bakar.
- Parse aşamasında kullanılan nesnelerin tanımları değişmesine izin vermemek için kilitlenirler.
- Kullanıcıların referans edilen şema nesnelerine erişim yetkisini kontrol eder.
- İfadenin optimum uygulama planı (execution plan) belirlenir.

- İfade ve uygulama planı paylaşılan SQL bölgesine yüklenir.

Parse aşaması gerçekleşmesi için gereken hazırlıkları yapar tekrar aynı ifadenin işlenmesinde bu aşama tekrarlanmaz. Oracle sunucu her bir SQL ifadesini yalnızca bir kere çevirir ve sonraki ifadelerde referans olur. Bununla beraber Oracle sunucu her uygulamada kullanıcının yetkisini kontrol eder.

Parse safhasında ifadenin geçerlilik kontrolünün yapılmasına rağmen yalnızca betimleme hataları tespit edilir. Bu sebeple bazı hatalar bu aşamada yakalanamayabilir. Örneğin, veri aktarmadaki hatalar ya da verideki hatalar ( örneğin bir anahtara çift kayıt atama) uygulama esnasında ortaya çıkar.

**SELECT ifadesinin uygulanması**

Bu aşamada Oracle sunucu gerekli bütün bilgilere ve kaynaklara sahip durumdadır, ifade uygulamaya geçilir ve sunucu proses veriyi ele geçirmek için hazırlanır.

**Sorgu satırlarının alınması**

Alım aşamasında satırlar gerekirse belirtilen sırada seçilir ve sunucu tarafından kullanıcıya gönderilir. Gönderilen satır sayısına bağlı olarak birden fazla alım işlemi yapılarak sorgu sonucu kullanıcıya transfer edilir.

### 3.1.2.2 Paylaşım havuzu bileşenleri

Parse aşaması sırasında sunucu proses SGA da paylaşım havuzu denilen bölgede SQL ifadesini compile eder. Paylaşım havuzunun büyüklüğü SHARED_POOL_SIZE ile initialization parametrelerinde belirtilir. Paylaşım havuzunun iki temel bileşeni vardır.

- Kütüphane cache (Library cache)
- Veri sözlüğü cache (Data dictionary cache)

**Library cache**

SQL paylaşım alanında en son kullanılan SQL ifadelerinin bilgileri library cache de saklanır. Paylaşımlı SQL bölgesi şu bilgileri içerir.

- SQL ifadesinin metnini
- Parse tree : İfadenin compile edilmiş versiyonu.
- Uygulama Planı: İfade execute edilirken uygulanacak adımları içerir.

Bir SQL ifadesi tekrar execute edilirse ve SQL paylaşım alanı ifadenin execute planını içerirse, sunucu ifadeyi tekrar parse etmeye ihtiyaç duymaz. Library cache, parse zamanını azaltığı ve bellek ihtiyacını düşürdüğü için uygulamaların performansını artırır.

**Data dictionary cache**

Bu alan sözlük cache ya da satır cache olarak da adlandırılır. Veritabanın en son sıklıkla kullanılmış olduğu veritabanı tanımlamalarının kümesidir. Veritabanı dosyaları, tablolar, indeksler, kolonlar, kullanıcılar, yetkiler ve diğer veritabanı nesnelerinin bilgilerini içerir.

Parse safhasında sunucu proses sözlük cache' i SQL ifadesindeki nesnenin adını doğrulamak ve kullanıcı yetkilerini kontrol etmek amacıyla kullanır.

**Database buffer cache**

Bir sorgu işlendiğinde, sunucu proses veritabanı buffer cache'de ihtiyaç duyulan bloğu arar. Eğer veritabanı buffer cache'de blok bulunamazsa sunucu proses blokları veri dosyasından okur ve buffer cache'a bir kopyasını yerleştirir. Böylelikle sonraki istekler fiziksel okumaya gerek olmadan cache den okunabilir.

Veritabanı buffer cache'in büyüklüğü Oracle blok büyüklüğüne eşittir ve DB_BLOCK_SIZE parametresi ile belirtilir. Buffer'ın sayısı DB_BLOCK_BUFFERS parametresinin değerine eşittir.

### 3.1.2.3 Program global alanı

Program Global Alanı (Program Global Area – PGA) ya da diğer adıyla Proses Global Alanı, bir sunucu proses ya da arka alan prosesi için veri ve kontrol bilgileri içeren bir bellek bölgesidir. PGA, SGA'nın birçok proses tarafından paylaşılmasının tersine yalnızca bir proses tarafından kullanılır.[1] PGA bir proses yaratıldığında işgal edilir ve proses sonlandığında boşaltılır. Bir dedicated sunucu konfigürasyonunda sunucu'ın PGA'ı şu bileşenleri içerir.

- Sıralama Alanı (Sort Area) : SQL ifadesini işlemede ihtiyaç duyulan sıralamalar için kullanılır.
- Oturum Bilgisi (Session Information) : Kullanıcı yetkilerini ve oturumun performans

istatistiklerini içerir.

- İmleç Durumu (Cursor State) : SQL ifadesinin işlenmesinde aşamasını gösterir.
- Yığın Alanı (Stack Space) : Diğer oturumların değişkenlerini içerir.

**DML ifadelerinin prosesi**

DML ifadesinin işlenmesi iki evreden oluşur;

- Parse sorgulama prosesindeki safha ile aynıdır.
- Uygulama sırasında veri değişiklikleri yapmak için ek proseslere ihtiyaç duyar.

DML ifadesinin uygulanma aşamaları :

1) Eğer veri ve rollback bloklar buffer cache de değilse, sunucu proses bunları veri dosyasından buffer cache'e okuyarak alır.

2) Sunucu proses işleme girecek satırları kilitler.

3) Sunucu proses redo log buffer'a, rollback'lere ve verilere yapılan değişiklikleri kaydeder.

   - Rollback blokları verilerin değiştilmeden önceki değerlerini kaydeder. Rollback blokları verinin önceki değerini, DML ifadesinin roll back edilmesine karşın saklarlar.
   - Veri blokları kaydın yeni değeri ile değiştirilir.

4) Sunucu proses rollback bloklarına kaydın DML den önceki değerini kaydeder ve veri bloklarını günceller. Her iki değişiklik de veritabanı buffer cache de yapılır. Buffer cache d değişiklik yapılan tüm bloklar kiri blok olarak işaretlenir ki bunun anlamı diskde ilişkili olduğu blokla artık aynı olmadığıdır.

### 3.1.2.4 Yeniden yap kütüğü

SGA nın bir bölümü olan bu buffer alanına sunucu prosesin veri dosyası bloklarına yapılan değişiklikler kaydedilir. Yeniden yap kütüğü (redo log buffer) aşağıdaki özelliklere sahiptir.

- LOG_BUFFER parametresi ile byte cinsinden büyüklüğü tanımlanır.
- Değiştirilen blokları, değişimin yerini ve yeni değerini bloklara kaydeder.
- Redo log buffer sırasıyla kullanılır.
- Sirküler bir buffer'dır, dolmasından sonra ve tüm eski redo girişleri redo log

dosyalarına kaydedildikten sonra tekrar kullanılır.

### 3.1.2.5 Yapılandırma parçaları

Sunucu proses bir değişiklik yapılmadan önce eski veri değerlerini yapılandırma parçalarına (rollback segment) kaydederler. Önceki değer aşağıdaki işlemlerde kullanılır,

- Eğer transaction roll back edilirse değişimleri geri almak için kullanılır.
- Henüz commit edilmemiş değişiklikleri diğer transactionlar görmeyeceği için roll back segmentler kullanılır böylelikle okuma tutarlığı sağlanmış olur.
- Veritabanının herhangi bir bozulması durumda iyileştirmek için.

Rollback segmentler tablolar ve indeksler gibidir, veri dosyalarında bulunur ve rollback bloklar ihtiyaç duyulmaları durumunda veritabanı buffer cache'e getirilirler.

Rollback segmentler DBA tarafından yaratılır. Rollback segmente yapılan değişiklikler redo log buffer'a kaydedilir.

**Onaylama işlemi**

Oracle sunucu hızlı commit mekanizmasını kullanarak instance daki herhangi bir aksaklıkda commit edilen değişikliklerin iyileştirilmesini garanti eder.

Bir transaction commit edilirse Oracle sunucu bu transaction'a bir commit system change number (SCN) atar. SCN artan değerdedir ve veritabanında tekdir. Bir commit işleminde,

1) LGWR redo log buffer'ı redo log dosyasına yazar. Bu noktadan sonra Oracle sunucu herhangi bir aksaklıkta veri kaybetmeyeceğini garanti etmiş olur.

2) COMMIT tamamlanınca kullanıcı bilgilendirilir.

3) Sunucu proses transaction'ın tamamlandığını ve kaynakların serbest bırakıldığı belirtmek için bilgileri kaydeder.

**Diğer işlemler**

**Log Yazıcısı (LGWR)**

Aşağıdaki durumlarda redo log buffer dan redo log dosyalarına sıralı olarak yazar,

- Bir transaction commit olunca,
- Redo log buffer'ın üçte biri dolunca,
- Redo log bufferda bir megabyte'ın üzerinde kayıt olunca,

- DBW0 bloklardaki değişiklikleri veritabanı buffer dan veri dosyalarına yazmadan önce.

**Veritabanı Yazıcısı (DBW0)**

Bir sunucu prosesi değişiklikleri rollbackleri ve buffer cachedeki veri bloklarınıdeğiştirir. Veritabanı yazıcısı buradaki kirli buffer'ı veritabanı buffer cache den veri dosyalarına yazar. Gereken sayıdaki tampon bölgesinin boş olmasını sağlar çünkü tampon bölgelerine boş olmadıkları sürece yazılamaz. Böylelikle performans artırılmış olur çünkü sunucu prosesleri yalnızca veritabanı tampon bölgesinde değişiklik yaparlar. Veritabanı yazıcıları aşağıdaki durumlarda veri dosyasına yazarlar;

- Her üç saniyede bir.
- Kirli tampon dolduğu zaman.
- Bir prosesin boş bir tampon bulamadığı zaman.
- Veri dosyası ile buffer cache bire bir doldurulduğu zaman.

**Sistem Monitör (SMON)**

Eğer bir instance başarısız olursa ve SGA da diske kaydedilmeyen bilgiler kaybolur. Instance kaybından sonra, veritabanı açılırken iyileştirme esnasında sistem monitör prosesleri görev alırlar. SMON prosesleri online redo dosyalardaki verileri okurlar ve değişikleri veri bloklarına yansıtırlar.

## 3.2 Veri Bütünlüğü

### 3.2.1 Veri bütünlüğü tanımı

Verilerin bütünlüğü veritabanı yöneticisi veya uygulama geliştiricisi tarafından tanımlanan bir dizi kural ile sağlanmaktadır. Veri bütünlüğü (data integrity) veritabanındaki verilerin aralarındaki uyumdur (Oracle, 1999b; s28-2). Uyumun sağlanması için kurallar oluşturulur.

Veri bütünlüğüne örnek olarak Envanter_Table ile Talep_Detay_Table tablolarını ele alalım.

Envanter_Table : Tablodaki Env_Kod tekildir.

Talep_Detay_Table : Bu tabloda her bir talebin ayrıntısı yer almaktadır. Talep_No kolonu tablo için tekildir. Durum kodu kolonu (O,B,R) değerleri dışında başka bir değer alamaz.

Miktar kolonuna yalnızca tam sayı değerleri girilebilir. Env_Kod ise Envanter_Table daki değerler dışında bir değer alamaz.

Çizelge 3.1 Tablolar arası ilişki

| Env_Kod | Env_Ad | Birim |
|---|---|---|
| 10 | Hesap Makinesi | Adet |
| 20 | Kalem | Düzine |

| Talep_No | Env_Kod | Miktar | Durum_Kod |
|---|---|---|---|
| 100 | 10 | 25 | O |
| 101 | 20 | 105 | R |
| 102 | 10 | 20 | B |
| 103 | 20 | 1 | O |
| 101 | 20 | 105 | |

Bu örnekte dikkat edilmesi gereken husus tablo verileri için bazı kolonlara belirli kısıtlamalar tanımlanabilir.

### 3.2.2 Veri bütünlüğü tipleri

- Null : Null kısıtlaması kolonun yeni giriş (insert) ya da güncelleme (update) işlemleri esnasında null (değerinin olmaması) ya da not null (dolu) olmasına izin verir.
- Tekil Kolon Değerleri: Tekil değerli tanımlanan bir kolon (ya da bir dizi kolon) satırın yeni girişi ya da güncellenmesi işleminde sadece tekil değer almasına izin verir. Yani tabloda ancak bir tane bulunabilmektedir ve null olabilirler.
- Ana Anahtar : Ana anahtar olarak tanımlanan bir anahtar ( bir kolon ya da bir dizi kolon) yeni girişi ya da güncellenmesi esnasında tekilliği sağlar. Ana anahtar tabloda tekildir ve anahtarı oluşturan kolon veya kolonlar null olamazlar.
- Referans Bütünlüğü: Referans bütünlüğü, tabloda tanımlanan bir anahtar (kolon ya da kolonlar) başka bir tabloda bulunan bir başka bir anahtarla eşleşmesini sağlar. Referans

bütünlüğü ayrıca referans edilen değer üzerinde hangi işlemlerin yapılabileceğinin ya da bu işlemlerin bağlı olduğu değeri nasıl etkileyeceğinin belirtilmesini de sağlar. Bazı referans kısıtlamaları;

- Sınırlama (Restrict) : Referans edilen verini değiştirilmesine ve silmesine izin vermez.
- Set to null : Eğer referans edilen veri güncellenir veya silinirse, ona bağlı bütün veriler null olur.
- Set to default : Eğer referans edilen veri güncellenir veya silinirse, ona bağlı bütün verilere varsayılan değer atanır.
- Cascade : Eğer referans edilen veri güncellenirse ona bağlı veriler de güncellenir ya da referans edilen veri silinirse ona bağlı veriler de silinir.
- No Action : Referans edilen verinin güncellenmesine ve silinmesine izin vermez. Oracle bunu varsayılan değer olarak atamıştır.

- Kompleks Bütünlük Kontrolü: Kompleks bütünlük kontrolü bir kolon için kullanıcı tanımlı bir kurallar sayesinde yapılır. Bu kurallar satırların yeni girişinde, güncellemede ve silmede kolonların değerlerine kısıtlamalar getirir.

### 3.2.3 Oracle'da veri bütünlüğü sağlama

Oracle yukarıda anlatılan kısıtlama kurallarını, bütünlük kısıtlamalarıyla (integrity constraints) ve veritabanı tetikleyicileriyle (database triggers) sağlar.

#### 3.2.3.1 Bütünlük kısıtlamaları

Bütünlük kısıtlaması (Integrity Constraint) tablo kolonlarını tanımlarken kullanılan bir metottur. Oracle aşağıdaki bütünlük kısıtlamalarını destekler.

- NOT NULL; kısıtlaması kolunun null olmamasını sağlar.
- TEKİL ANAHTAR (UNIQUE KEY); kısıtlaması kuralına bağlı olan kolonların tekil değerler alması için kısıtlar.
- ANA ANAHTAR (PRIMARY KEY); kısıtlaması kuralına bağlı olan kolonların ana anahtar değerler alması için kısıtlar.
- YABANCI ANAHTAR (FOREIGN KEY); kısıtlaması bağlı olduğu kolonları referans bütünlüğüne zorlar. Oracle aşağıdaki durumlardaki FOREIGN KEY bütünlük kısıtlamalarına destek verir.
  - Update ya da delete NO Action
  - Delete CASCADE

- Delete SET NULL

- KONTROL (CHECK) kısıtlaması kompleks bütünlük kısıtlama kurallarındandır.

### 3.2.3.2 Veritabanı tetikletiyicisi

Oracle bütünlük kurallarını veritabanı triggerlarını kullanarak da sağlar. Bu triggerlar veritabanına gömülü prosedürler olup, yeni giriş, güncelleme ve silme operasyonlarında aktif hale geçerler. Bölüm 4.1'de tetiklere değinilmiştir.

Bütünlük Kısıtlamalarının Avantajları

- Veritabanı uygulamasının koduna kuralları uygulatır,
- Gömülü prosedürleri kullanarak veriye erişimi kontrol altında tutar.
- Kuralları gömülü prosedür triggerlarıyla uygulatır.
- Bütünlük kısıtlamalarının tanımlanması çok kolaydır. Ek bir programlamaya ihtiyaç duymadan SQL ile tanımlanır. Hataları ayıklamak da bir o kadar kolaydır.

**Tablo kısıtlamaları**

Table_constraint tanımlama tablonun tanımlanmasının bir parçasıdır. Bir bütünlük kısıtlaması kuralları tablonun kolon veya kolonlarına uygulanır. CREATE TABLE ya da ALTER TABLE ifadelerinde yer alabilirler.

**Kolon kısıtlamaları**

Kolon kısıtlamaları kolon tanımlamalarının bir parçası olarak kullanılır.

- CREATE TABLE ya da ALTER TABLE ADD ifadelerinde her çeşit bütünlük kısıtlaması tanımlanabilir.
- ALTER TABLE MODIFY ifadesiyle yalnızca NOT NULL kısıtlaması eklenebilir ya da kaldırılabilir.

Kısıtlamaları Oracle isimleri ve tanımlamalarıyla veri sözlüğünde saklar. Eğer kısıda isim verilmezse Oracle SYS_C*n* formatında bir isim verir.

Eğer kolonun null olup olmadığı belirtilmezse Oracle varsayılan olarak kolonun null olabilmesini sağlar. Null kısıtlaması REF ve LOB tipindeki kullanıcı-tanımlı nesnelerde kullanılamaz.

**Tekil anahtar**

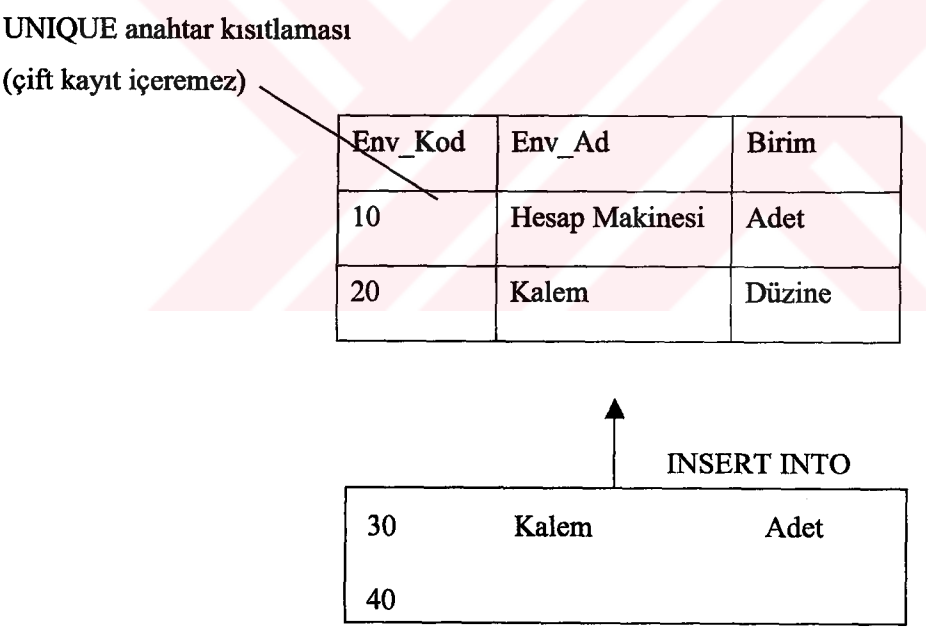
Tekil anahtar (Unique key), belirtilen kolon ya da kolonların tabloda tekil olmasını sağlar. Bunula beraber bir tek kolon da unique anahtary olabilir ve null değerler alabilir.

Bileşik unique anahtar kolonların kombinasyonundan oluşmaktadır. Anahtarı oluşturan kolonlar null olabilir ama bir tabloda anahtarı oluşturan kolonlar iki satırda da null olamaz. Çünkü tekillik özelliğine aykırı bir durum söz konusudur.

Kısıtlamaları:

- Tablodaki iki satır anahtarları aynı değer alamaz.
- Bir bileşik anahtar otuz iki kolondan fazla kolon içeremez.
- Tekil anahtarı oluşturan kolonlar LONG ve LONG RAW tipinde olamaz.
- Aynı kolon ya da kolonlar tablonun hem ana anahtarı hem de unique anahtarı olamaz.

Çizelge 3.2 Tekil anahtar kısıtlaması




Örnekteki ilk kayıt kaydedilemez çünkü unique anahtar kısıtlaması olan bir kolona aynı değerli bir kayıt eklenemez. Diğer kayıdın kolonu boş olduğu için kısıtlamaya aykırı bir durum ortaya çıkmaz.

**Ana anahtar**

Bir tablonun bir kolon ya da kolonlarının kombinasyonu tablonun ana anahtarı olabilir. Bileşik ana anahtar (Composite Primary Key) kolonların kombşnasyonundan oluşur.

Kısıtlamaları:

- Bir tablonu yalnız bir tane ana anahtarı vardır.
- Ana anahtardaki kolonlar LONG ve LONG RAW veri tipinde olamaz.
- Ana anahtar değeri tabloda bir kez yer alır, başka bir satırın da değeri olamaz.
- Ana anahtarında kolonları null olamazlar.
- Ana anahtarın indeks-organizeli tablosunun büyüklüğü veritabanının veri blok büyüklüğünün bir buçuk katını ya da 3800 byte değerini geçemez.
- Bileşik ana anahtar asla otuz iki kolondan fazla kolon içeremez.
- Aynı kolon ya da kolonlar tablonun hem ana anahtarı hem de unique anahtarı olamaz.

Çizelge 3.3 Ana anahtar kısıtlaması

| Env_Kod | Env_Ad | Birim |
|---|---|---|
| 10 | Hesap Makinesi | Adet |
| 20 | Kalem | Düzine |

INSERT INTO

| | | |
|---|---|---|
| 30 | Defter | Adet |
| | Masa | Adet |

Örnekte Envanter_Table tablosunun ana anahtarı env_kod olarak belirtilmiştir. Yeni giriş yapılacak satırlardan ilki tekilliği bozmadığı için kaydedilir ama diğer satırda env_kod null olduğundan ana anahtar kısıtlaması sayesinde kaydedilemez.

**Not null kısıtlaması**

Bir kolon kısıtlamasıdır, tablo kısıtlamalarında belirtilemezler. Not null kısıtlaması kolonun null olamamasını sağlar. Null kısıtlaması tanımlanmasına gerek yoktur, herhangi bir null kısıtlama içermeyen kolon varsayılan olarak null olabilirler.

Çizelge 3.4 Not null örneği

| Talep_No | Env_Kod | Miktar | Durum_Kod |
|---|---|---|---|
| 100 | 10 | 25 | O |
| 101 | 20 | 105 | R |

NOT NULL KISITLAMASI (tabloda bu kolon asla null olamaz.)

**Referans bütünlük kısıtlamaları**

Tablonun kolon ya da kolonlarını yabancı anahtar (foreign key) olarak belirleyen bu kısıtlama türü bu yabancı anahtarla başka bir tablodaki ana ya da unique anahtarlar arasında bir ilişki kurar. Diğer tabloda referans edilen temel ya da unique anahtarlara referans edilen anahtar (referenced key) denir. Yabancı anahtarı içen tabloya çocuk tablo (child table), referans edilen anahtarı içen tabloya da baba tablo (parent table) adı verilir. Yabancı anahtarla referaans edilen anahtar aynı tabloda yer alabilir. Bu durumda parent tablo ile child tablo aynı tablo olur. Bir tablonun ana anahtarı ya da unique anahtarı aynı zamanda yabancı anahtar da olabilir.

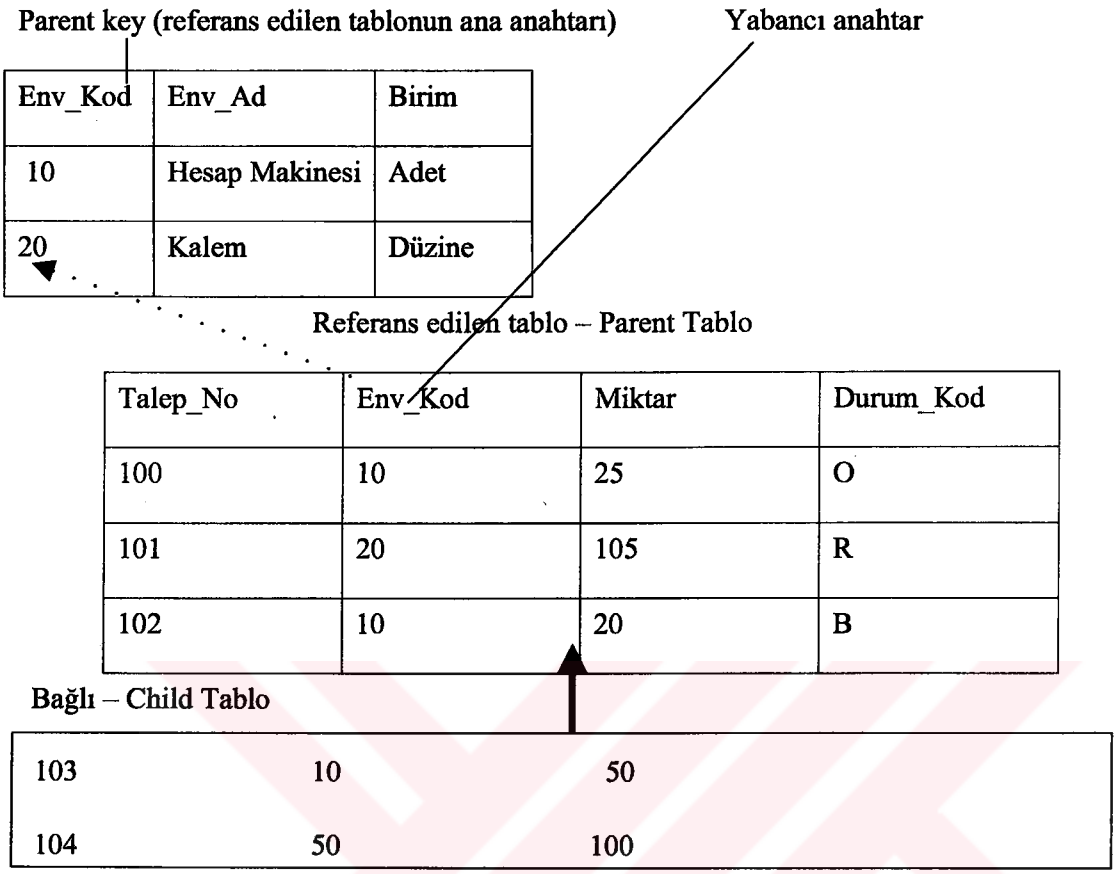
Bir tabloda birden çok yabancı anahtar yer alabilir. Aynı zamanda bir kolon birden fazla yabancı anahtarın parçası olabilir.

Kısıtlamaları:

- Yabancı anahtar LONG veya LONG RAW veri tipinde olamaz.
- Referans edilen ana anahtar ya da unique anahtar parent tabloda tanımlı olam zorundadır.
- Parent ve child tabloları aynı veritabanı üzerinde olmalıdır. Ayrık veritabanlarını referans gösterebilmesi için veritabanı triggerları kullanılır.

Çizelge 3.5'de envanter_table tablosunun env_kod kolonu primary anahtardır. Bu kolon Talep_Detay_Table tablosunun env_kod kolonuna referans edilmiştir. Talep_Detay_Table tablosunun env_kod kolonu bir yabancı anahtar olmuştur. Bu durumda Talep_Detay_Table tablosu ise child table, Envanter_Table tablosu da parent table olmuştur. Örnekte yeni girişi yapılan ilk kayıt, yabancı anahtar olarak 10 nolu envanter_table da tanımlı olan bir envanter kodu içerir. Oysaki diğer kayıt envanter tablosunda kayıtlı olmayan 50 kodunu içerdiği için yeni girişi yapılamaz.

Çizelge 3.5 Referans bütünlük kısıtlaması




**Kontrol kısıtlaması**

Tablonun herbir satırı için kolon veya kolonlara verilen kısıtlamalar. Check kısıtlamaları aşağıdaki yapıları içermez.

- Başka bir satırdaki değerleri referans eden sorgular,
- SYSDATE, UID, USER ya da USERENV fonksiyonlarını çağırılması,
- CURRVAL, NEXTVAL, LEVEL ya da ROWNUM sahte kolonları (pseudocolumns),
- Belirsiz veri sabitleri.

Kısıtlamaları Durumları

- ENABLE kısıtlamayı geçerli kılari
- DISABLE kısıtlamanın geçerliğini kaldırır.
- VALIDATE tabloda varolan verilerin kısıtlamaya uygunluğunu kontrol eder.
- NOVALIDATE tablodaki bazı verilerin kısıtlamaya uygun olamayabileceği durumdur.

Ayrıca;

- ENABLE VALIDATE , enable ile aynıdır.
- ENABLE NOVALIDATE kısıtlamanın geçerli olduğunu ama tüm satırla için doğru olmak zorunda olmadığı belirtir.
- DISABLE NOVALIDATE, disable ile aynı anlamdadır. Kısıtlama kontrol edilmez ve doğrulanması da şart değildir.
- DISABLE VALIDATE kısıtlamaların geçerli olmadığı durumdur. Kısıtlama üzerinedeki tüm indeksler kaldırılır ve kısıtlanmış kolonların modifikasyonuna izin vermez.

## 3.3 Veritabanı Kullanıcıları

Veritabanı yöneticisi veritabanına erişime izin vermek için kullanıcı isimleri yaratırlar. Bir kullanıcı veritabanına erişirken bazı mekanizmalar tarafından doğrulanırlar. Bu mekanizmalar;

- Veritabanı
- İşletim sistemi
- Ağ

### 3.3.1.1 Kullanıcı yaratma

```
CREATE USER user
IDENTIFIED {BY password | EXTERNALLY}
[TEMPORARY TABLESPACE tablespace]
[QUTO {interger [ K | M] | UNLIMITED }} ON tablespace
 [QUTO {interger [ K | M] | UNLIMITED }} ON tablespace ] ... ]
[PASSWORD EXPIRE]
[ACCOUNT { LOCK | UNLOCK } ]
[PROFILE { profile | DEFAULT }]
```

user: kullanıcı ismidir.

BY password: kullanıcının veritabanına bağlanmasında kullanılacak şifresi belirtilir.

TEMPORARY TABLESPACE: kullanıcının kullanacağı tablespace belirtilir.

QUTO: yukarda belirtilen tablespace üzerinde kullanıcının hakkı olan maksimum alanı belirtir. Interger olarak byte ya da kilobyte cinsinden tanımlanır.

PASSWORD EXPIRE:kullanıcıya veritabanına bağlandığında şifresini değiştirmeye zorlar.

ACCOUNT LOCK/UNLOCK:kullanıcının hesabını kilitler ya da kiliti bozar. Varsayılan değeri UNLOCK dır.

PROFILE: kontrol kaynak kullanımında kullanılır.

**İşletim sisteminde kullanıcı yaratma:**

Bu metod Oracle Server üzerinden kullanıcı bağlanması esnasında kullanılırsa daha faydalı olur. CREATE USER komutu ile IDENTIFIED EXTERNALLY seçimi belirtilirse bu kullanıcı işletim sistemi tarafından da kontrol edilir.

## 3.4 Haklar, Roller Ve Güvenlik Politikası

İmtiyaz, belirli bir tip SQL ifadesini execute etmek ya da başka bir kullanıcını nesnesine erişmek için kullanıcılara verilen haklara denir. Bazı imtiyazlar;

- Veritabanına bağlanma (oturum açma)
- Tablo yaratma
- Diğer kullanıcıların nesnelerinden sorgulama
- Diğer kullanıcıların gömülü prosedürlerini kullanabilme.

Kullanıcılara verilen imtiyazlar onların görevlerini başarıyla tamamlamalarını sağlar. İmtiyazlar verilirken gerçek anlamda ihtiyaç duyan kullanıcılara verilmelidir. Bir kullanıcı iki yolla imtiyaz kazanabilir.

- Hak direk olarak kullanıcılara verilebilir. Örneğin STOK kullanıcısına envanter_table tablosuna yeni giriş hakkı verilebilir.
- Haklar roller adı verilen haklar grubuna verilebilir. Örneğin MEMUR isimli bir role envanter_tablosuna yeni giriş hakkı verilir ve rol STOK kullanıcısına grant edilir.

İki tip imtiyaz vardır:

1) Sistem imtiyazı

2) Şema nesneleri imtiyazı

### 3.4.1 Sistem imtiyazları

Bir sistem imtiyazı veritabanındaki belirli bir harekete ya da belirli bir tip şema nesne üzerindeki bir harekete verilen haklara denir. Örneğin tablo yaratımı ya da tablodaki satırların silme hakları sistem imtiyazıdır. Almışın üzerinde sistem imtiyazı vardır.

#### 3.4.1.1 Sistem imtiyazlarını verilmesi ve geri alınması

Sistem imtiyazları kullanıcılara ve rollere verilebilir ya da geri alınabilir. Bu işlemler SQL komutlarıyla ya da Oracle Enterprise Manager dan yapılabilir.

```
GRANT {system_imtiyazı | rol}
        [,{system_imtiyazı | rol}] ...
   TO  {kullanıcı | rol | PUBLIC}
        [,{kullanıcı | rol | PUBLIC} ] ...
   [WITH ADMIN OPTIONS]
```

system_imtiyazı : verilecek sistem imtiyazıdır.

rol: kullanım hakkı verilecek roldür.

PUBLIC:sistem imtiyazını tüm kullanıcılara verir.

WITH ADMIN OPTIONS: verilen hakları diğer kullanıcılara da verebilme opsiyonudur. Yani bir sistem imtiyazını GRANT edebilmek için WITH ADMIN OPTIONS opsiyonuna ihtiyaç vardır. Ayrıca hakkı geri almak için de WITH ADMIN OPTIONS opsiyonuna ihtiyaç vardır.

GRANT ANY ROLE sistem imtiyazına sahip olan kullanıcı veritabanındaki her rolü grant edebilir.

DBA_SYS_PRIVS veritabanındaki tüm imtiyazları içerir. SESSION_PRIVS ise o anki oturumun sahip olduğu imtiyazları içerir.

Ancak grant edilmiş imtiyazlar revoke ile geri alınabilir.

```
REVOKE {system_imtiyazı | rol}
        [,{system_imtiyazı | rol}] ...
   FROM  {kullanıcı | rol | PUBLIC}
```

[,{kullanıcı | rol | PUBLIC} ] ...

[WITH ADMIN OPTIONS]

### 3.4.2 Şema nesneleri imtiyazı

Şema nesnesi imtiyazı belirli bir tabloya, görüntüye, sıraya, prosedüre ya da pakete uygulanacak işlemlere verilen haklardır. Farklı şema nesneleri için farklı imtiyazlar bulunmaktadır. Örneğin, envanter tablosundan bir satır silmek bir nesne imtiyazıdır.

Her şema nesnesinin şema imtiyazı yoktur, bu nesneler sistem imtiyazlarını kullanırlar. Örneğin cluster, indeksler, triggerlar, veritabanı bağlantılarıdır. ALTER CLUSTER için kullanıcının ALTER ANY CLUSTER sistem imtiyazına hakkı olması gerekmektedir. Çizelge 3.6'da nesnelerin sahip olduğu imtiyazlar belirtilmiştir.

Çizelge 3.6 Nesne imtiyazları

| Nesne İmtiy. | Tablo | Görüntü | Sıra | Prosedür |
|---|---|---|---|---|
| ALTER | X | | X | |
| DELETE | X | X | | |
| EXECUTE | | | | X |
| INDEX | X | | | |
| INSERT | X | X | | |
| REFERENCES | X | | | |
| SELECT | X | X | X | |
| UPDATE | X | X | | |

GRANT {nesne_imtiyazı [(kolon_list)]

,nesne_imtiyazı [(kolon_list)] ] ...

| ALL [PRIVILEGES]}

ON [schema.]oblect

```
TO {kullanıcı | rol | PUBLIC}
    [,{kullanıcı | rol | PUBLIC} ] ...
[WITH GRANT OPTIONS]
```

nesne_imtiyazı: kullanım hakkı verilecek imtiyaz.

kolon_list: INSERT, REFERENCES ya da UPDATE imtiyazı verilen bir tablonun veya görüntünün kolonu belirtir.

ALL:tüm imtiyazları WITH GRANT OPTION ile verilir.

ON nesne:imtiyaz hakkı verilecek nesnedir.

WITH GRANT OPTION: nesne imtiyazlarını diğer kullanıcılara verebilme opsiyonunu tanır.

DBA_TAB_PRIVS tablosu veritabanındaki tüm nesne imtiyazları içerir.

```
REVOKE {nesne_imtiyazı
        [,nesne_imtiyazı ] ...
      | ALL [PRIVILEGES]}
  ON [schema.]oblect
  FROM {kullanıcı | rol | PUBLIC}
      [,{kullanıcı | rol | PUBLIC} ] ...
[CASCADE CONSTRAINT]
```

Grant komutundan farklı olan parametreleri:

FROM :hakkın alınacağı kullanıcı veya rolden önce kullanılır.

CASCADE CONSTRAINT: REFERENCES ya da ALL imtiyazlarıyla tanımlanan referans bütünlük kısıtlamalarının hepsini siler.

### 3.4.3 Roller

Roller birbiriyle bağlantılı imtiyazların oluşturduğu gruplardır. Roller sayesinde Oracle imtiyazları kontrol altında tutarak daha kolay yönetimini sağlar.

- İmtiyazların grant edilme sayılarını azaltır. Bir dizi imtiyazı kullanıcılara tek tek

vermektense, imtiyazları içeren bir rol yaratılıp, bu rolün kullanıcılara grant edilmesi daha sadeleşmiş bir işlemdir.

- Dinamik imtiyaz yönetimine olanak sağlar. Şöyle ki bir role bağlı bir imtiyazın modifiye edilmesiyle bu role sahip tüm kullanıcılar otomatik olarak ve derhal modifikasyona sahip olurlar.
- Roller kullanıma açılarak ya da kapanarak imtiyazlar da geçici olarak kullanım dışı bırakılabilir.
- İşletim sistemi sayesinde komut ya da yardımcı programlarla veritabanındaki kullanıcılara roller atanabilir.
- Roller kullanarak veri sözlüğünde daha az sayıda grant saklanır böylelikle performans artırılmış olur.

#### 3.4.3.1 Rol yaratma

Yazılım dizisi:

```
CREATE ROLE role [NOT  IDENTIFIED | IDENTIFIED
    {BY password | EXTERNALLY} ]
```

role: rolün adıdır.

NOT IDENTIFIED: rol kulanılabilir olduğunda doğrulanmaya ihtiyaç duymayacağını yani şifreye gereksinim olmadığını belirtir.

IDENTIFIED: rol kulanılabilir olduğunda doğrulanmaya ihtiyaç duyacağını belirtir.

BY password: rol kullanıldığında kullanıcını vermesi gereken şifredir.

EXTERNALLY: rolü kullanmadan önce veritabanının dışındaki bir servis aracılığı (örneğin işletim sistemi) ile kontrol edilir.

#### 3.4.3.2 Rol değişimi

Yazılım dizisi:

```
ALTER ROLE role [NOT  IDENTIFIED | IDENTIFIED
    {BY password | EXTERNALLY} ] ;
```

#### 3.4.3.3 Kullanıcıya rol verme

```
GRANT ROLE role [,role ] ..
```

TO {user | role | PUBLIC}

[,{user | role | PUBLIC} ] ...

[ WITH ADMIN OPTION ]

#### 3.4.3.4 Kullanıcıdan rol geri alma

REVOKE ROLE role [,role ] ..

FROM {user | role | PUBLIC}

[,{user | role | PUBLIC} ] ...

# 4. PL/SQL VE TETİKLER

## 4.1 PL/SQL

PL/SQL (PROCEDURAL LANGUAGE / SQL), Oracle'ın prosedürel dillere ait özellikleri SQL'e eklemesiyle oluşan dördüncü kuşak bir programlama dilidir (Oracle, 1999f).

Özellikleri;

- PL/SQL blok yapılı bir dildir. PL/SQL'i oluşturan temel parçalar (prosedürler, fonksiyonlar, anonim bloklar) iç içe alt bloklardan oluşmuş mantıksal bloklardır. Her bir mantıksal blok bir problemi ya da alt problemi çözmek için tasarlanmıştır.
- Akış kontrolünü sağlamak için yapısal dillerin kullandığı şart cümlelerini ve döngüleri kullanır. Bu özellik sayesinde programlar daha hızlı ve etkin bir yapıdadır.
- Performansı prosedürel dillerdeki gibi programların kısa olması ve hızlı çalışması performansı artırmaktadır.
- SQL'i destekler. Entegre olarak SQL'i kullanır. PL/SQL SQL'in tüm veri tiplerini ve NULL değerini destekler.
- Nesneye dayalı programlamayı destekler.
- Verimliliği kullandığı Oracle Forms ya da Oracle Reports gibi araçlar sayesinde artırır.
- Güvenlik,istemci uygulamalarının veritabanının genel yapısına zarar vermemesi için PL/SQL de veritabanı tetikleyicileri yazılır.

PL/SQL derlemesi ve çalışma zamanı sistemi bağımsız bir ürün değildir. Bu teknolojiyi bir makine PL/SQL blok ve alt bloklarını derler ve icra eder mantığıyla düşünebiliriz. Bu makine prosedürel ifadeleri icra eder ama SQL ifadelerini SQL ifadesi düzenleyicisine gönderir. Makinede ya Oracle sunucusu olabilir ya da uygulama geliştirici araçlar örneğin Oracle Forms ya da Oracle Reports gibi, yüklü olabilir.

### 4.1.1 Blok yapısı

Bir blok mantıksal olarak ilişkili grup tanımlamalarını ve ifadelerini içerir. Tanımlamalar blok içinde geçeridir, lokaldir ve blok bitince tanımlamalar geçersiz olur.

Bir PL/SQL bloğunu üç bölümden oluşur.

1) Tanımlama bölümü

2) İcra edilebilen (executable) edilebilen bölüm

3) İstisna yönetimi bölümü

PL/SQL programları içersinde kullanılan program parçalarıdır. Yapısı aşağıdaki gibidir;

```
DECLARE
-- tanımlamalar
BEGIN
-- ifadeler
[EXCEPTION
-- istisnalar
END;
```

Alt bloklar PL/SQL bloğun ya da alt programları icra ya da istisna bölümüne gömülebilirler ama tanımlama kısmında olamazlar. Aynı zamanda lokal alt programlar bloğun tanımlama kısmında tanımlanabilir. Böylece alt programlar tanımlandıkları bloklardan çağrılabilirler.

### 4.1.2 Alt programlar

1) Fonksiyonlar

2) Prosedürler

#### 4.1.2.1 Fonksiyonlar

Fonksiyonlar değer döndüren alt programlardır. Yapısı aşağıdaki gibidir;

```
FUNCTION fonksiyon_ismi
-- tanımlamalar
BEGIN
-- ifadeler
[EXCEPTION
-- istisnalar
END;
```

#### 4.1.2.2 Prosedürler

Prosedürler fonksiyonlardan farklı olarak birden çok değer döndürebilin alt programlardır. Yapısı aşağıdaki gibidir;

```
PROCEDURE prosedür_ismi
-- tanımlamalar
BEGIN
```

-- ifadeler

[EXCEPTION

-- istisnalar

END;

### 4.1.3 Değişkenler ve sabitler

PL/SQL, SQL'de ve prosedürel ifadeler kullanmak için sabit ve değişken tanımlamaya imkan tanır.

Değişkenler tüm CHAR, DATE, NUMBER gibi SQL veri tipini ya da BOOLEAN , BINARY_INTEGER gibi PL/SQL veri tiplerini kullanabilirler.

#### 4.1.3.1 Sayı tipleri

- Binary_integer: İşaretli tam sayı veri tipidir. Büyüklük aralığı –2147483647 .. 2147483647 sayıları arasındadır. PLS_INTEGER gibi BINARY_INTEGER da bellekte NUMBER veri tipinden daha az yer kaplar. Ama operasyonu PLS_INTEGER den daha yavaştır. Temel tip alt tiplerin türetildiği bir veri tipidir. BINARY_INTEGER'ın alt tipleri;
  - NATURAL
  - NATURALN
  - POSITIVE
  - POSITIVEN
  - SIGNTYPE
- Number: Nümerik veri tipidir. Yazılım dizisi;

NUMBER(tam,ondalık)

Tam sayının tüm basamaklarının toplam sayısıdır, ondalık ise noktanın sağında kalan basamak sayısıdır.

NUMBER(10,4) sabit-noktalı sayı tipine örnek teşkil eder.

NUMBER floating-point number dır.

NUMBER(10) ondalığı olmayan bir tam sayı dır.

NUMBER alt tipleri;

- DEC
- DECIMAL

- DOUBLE PRECISION
- FLOAT
- INTEGER
- INT
- NUMERIC
- REAL
- SAMMINT

- PLS_INTEGER: Bu veri tipleri işaretli tam sayıları saklamak için kullanılır. Alabileceği değer aralığı –2147483647 .. 2147483647 dir. Bellekte NUMBER dan daha az yer kaplar. BINARY_INTEGER ve PLS_INTEGER veri tiplerinin değer aralıklarının aynı olmasına rağmen tam olarak bire bir benzememektedirler. Eğer bir hesaplamada PLS_INTEGER değer aralığının dışına çıkarsa aykırı bir durum oluşur. Ancak BINARY_INTEGER veri tipiyle yapılan bir hesaplamada aşma NUMBER veri tipine karşılık geliyorsa aykırı bir durum ortaya çıkamz.

#### 4.1.3.2 Karakter tipleri

Krakter tipleri alfanümerik verileri saklamada, kelimeleri ve metinleri göstermede ve karakter stringlerle yapılan işlemlerle kullanılır.

- CHAR: Sabit uzunluklu karakter veri için kullanılır.

CHAR[(maximum_uzunluk)]

Maksimum uzunluk 1 .. 32767 arasında bir değer alabilir. Eğer uzunluk belirtilmezse 1 alınır. Verilen bu uzunluk değerleri karakter sayısı değil byte cinsindendir. Maksimum uzunluk 2000 byte olabilir.

- LONG ve LONG RAW : LONG veri tipi değişken uzunluktaki karakter stringler için kullanılır. LONG, VARCHAR2 gibi bir veri tipidir ama maksimum uzunluğu 32760 byte'tır. LONG RAW binary veri ya da byte string verileri için kullanılır. LONG veri tipi gibidir ama PL/SQL LONG RAW veri tipini yorumlamaz. LONG kolonları metin, karakter dizisi ve hatta kısa dökümanları saklar.
- RAW: Binary veri ve byte string saklamak için kullanılır. RAW verisi varchar2 tipi gibidir ama PL/SQL yorumlayamaz. Ayrıca bu veri tipinde değişkenin uzunluğu 37767 byte'a kadar belirtilebiliyor.

RAW(maksimum_uzunluk)

- ROWID ve UROWID : Her veritabanı tablosunun ROWID isimli ikili değerli bir sanal kolonu bulunur. Rowid'ler tablonun satırlarının depolanma adreslerini gösterirler. Fiziksel rowid tablodaki bir satırı belirtir. Mantıksal rowid ise index-organized tablosundaki bir satırı belirtir. ROWID yalnızca fiziksel rowidleri saklama kullanılırken, UROWID fiziksel, mantıksal ve yabancı rowidleri saklayabilir.

Bir rowid dört bölümden oluşur. OOOOOOFFFBBBBBBRRR

1) OOOOOO: Veri nesnesi sayısı veritabanı segmentini belirtir. Aynı segmentteki şema nesneleri aynı veri nesne sayısını alırlar.

2) FFF: Satırı içeren veri dosyasını belirten dosya numarasıdır. Dosya numaraları veritabanında tekildir.

3) BBBBBBB: Satırı içeren bloğun numarasıdır. Blok numaraları veri dosyaları ile ilintilidir, tablespacelerle değil. Yani aynı tablespacedeki iki satır farklı veri dosyalarında olsunlar, bunların aynı blok numaraları olabilir.

4) RRR: Bloktaki satırın numarasıdır.

- VARCHAR2: Değişken uzunluklu karakter verileni saklamada kullanılır. Uzunluğunu belirten bir parametre alır.

VARCHAR2(maksimum_uzunluk)

Maksimum uzunluğu 4000 byte olabilir.

- LOB: LARGE OBJECT veri tipleri yapılandırılmamış ve 4 gigabyte'ı geçmeyen veri bloklarını (örneğin imaj, video klips, ses dosyaları) saklamayı sağlarlar. LOB veri tipleri;
  - BFILE: İkili veri nesnelerini veritabanının dışına işletim sisteminde saklamaya yarar. Bu veri tipi dosya yerleştiricisini saklayarak server üzerideki büyük ikili dosyayı işaret eder. Bu yerleştiriciler, klasör diğer adı ve tam yolunu içerir. BFILE lar yalnızca okuma modunda açılırlar.
  - BLOB: bu veri tipleri veritabanındaki satır içi ya da satır dışı büyük ikili nesneleri saklarlar. Bu tipin de bir yerleştiricisi vardır ve nesneyi işaret ederler. Büyüklükleri 4 gigabyte'ı aşmaz.
  - CLOB
  - NCLOB

#### 4.1.3.3 Diğer tipler

- BOOLEAN: TRUE, FALSE ve NULL mantıksal değerlerinin saklanmasında kullanılan bir veri tipidir. Yalnızca mantıksal operasyonlarda kullanılabilirler. Parametre almaz yalnızca ilk değer olarak TRUE, FALSE veya NULL değerlerini alabilir. Veritabanı kolonuna TRUE ya da FALSE kaydedilemez.
- DATE: Sabit uzunluklu tarih/zaman değer tipidir. PL/SQL otomatik olarak karakteri varsayılan date tipine çevirebilir. Ayrıca aritmetik ifadelerde tarih güne çevrilerek işlem görür.

SYSDATE +1 gibi.

Varsayılan tarih formatı NLS_DATE_FORMAT parametresi ile belirtilir.

### 4.1.4 Kontrol yapıları

Her programlama dili temel kontrol yapılarını kullanarak yazılır.

- Seçim : Bir durumu test edikten sonra doğru ya da yanlış olmasına göre ifadeyi seçer ve işler. Durum Boolean (true/false) bir değer döndüren herhangi bir değişken veya ifade olabilir.
- Döngüler : İterasyonda bir dizi ifade durum doğruluğunu koruduğu sürece tekrarlanır.
- Sıralı: Sıralı yapılarda ifadeler bulundukları sıra itibariyle işlem görürler.

#### 4.1.4.1 Koşula bağlı kontrol

IF-THEN

IF ifadelerinin en basit yapısıdır. Eğer şart doğru ise ifadeler icra edilir.

```
IF şartlar THEN
  İfadeler
END IF;
```

IF-THEN-ELSE

Bu tip şart cümlelerinde şartların doğruluğuna alternatif olarak ELSE den sonra icra edilecek ifadeler bulunmaktadır.

```
IF şartlar THEN
  İfadeler1
```

```
ELSE
  İfadeler2
END IF;
```

IF-THEN-ELSIF

```
IF şartlar1 THEN
  İfadeler1
ELSIF şartlar2 THEN
  İfadeler2
ELSE
 İfadeler3
END IF;
```

Bu şart cümlerinde ilk doğrulanan şart ifadesi icra edilir ya da hiçbir doğrulanmazsa ELSE den sonraki ifadeler icra edilir. Örneğin;

```
IF tutar > 5000 THEN
  Komisyon := 10;
ELSIF tutar >  1000 THEN
  Komisyon :=  5;
ELSE
  Komisyon :=  2;
END IF;
```

Eğer tutar 5000 den büyükse birinci ve ikinci şart doğrulanmış olur ama komisyon 10 değerini alır. Çünkü birinci doğrulandıktan sonra ikinci test edilmez.

Bölüm 7'de Şekil 7.18'de bir if örneği yer almaktadır.

### 4.1.4.2 Döngü kontrolleri loop ve exit ifadeleri

**LOOP**

Döngüler herhangi bir EXIT ya da GOTO komutuna rastlamadığı sürece tekrarlanır. Bölüm 7'de form örneğinde loop döngü kontrolüne örnek verilmiştir.

```
LOOP
  ifadeler
END LOOP;
```

**EXIT**

EXIT komutu döngüden şartsız ve derhal çıkışı sağlar. Bir döngü içinde kullanılması şarttır.

```
LOOP
  İfadeler
  EXIT ...
END LOOP;
```

**EXIT-WHEN**

EXIT-WHEN bir döngünün şart bağlı olarak sonlamasını sağlar. Eğer WHEN den sonraki koşul doğru ise döngü tamamlanır ve bir sonraki ifadeye geçilir.

```
LOOP
  İfadeler
  EXIT  WHEN koşul
END LOOP;
```

**WHILE-LOOP**

WHILE-LOOP ifadesi bir koşula bağlı olarak döngüyü sağlar. Eğer WHILE dan sonraki koşul doğrulanmazsa döngüye son verilir.

```
WHILE koşul LOOP
 İfadeler ...
END LOOP;
```

**FOR-LOOP**

FOR belirli bir saqyıda tekrarlanan döngülerdir. İşlem öncesi döngünün kaç kere tekrarlanacağı bilinirse FOR-LOOP döngüsü kullanılır. Yapısı şu şekildedir;

```
FOR kontrol_değişkeni IN [REVERSE] <alt_sınır> .. <üst_sınır>  LOOP
 ifadeler
END LOOP;
```

- kontrol_değişkeni: döngünün her tekrarın azalan ya da artırılan değişkendir. Döngü başladığında otomatik olarak tanımlanır, bitiminde ise otomatik olarak iptal edilir.
- alt_sınır ve üst_sınır, döngünün kontrol değişkeninin alacağı değerlerin sayı aralığıdır.
- REVERSE: Döngü kontrol değişkeni büyükten küçüğe azalarak değer alacaksa REVERSE kullanılır, kullanılmazsa küçükten büyüğe değer alır.

### 4.1.4.3 Sıralı kontroller

## GOTO

GOTO ifadeleri program içinde tanımlanan etiketlere gitme işlemini sağlar. Etiketler <<etiket_adı>> şeklinde tanımlanır.

```
BEGIN
  .....
  <<basla>>
  BEGIN
    .....
  END;
  GOTO basla;
    ......
END ;
```

Sınırlamaları:

- IF ifadesinin içindeki bir etikete ifadenin dışından GOTO ile gönderimde bulunulamaz.
- Bir IF ifadesinin bir koşulundan ELSE kısmına GOTO ile gönderimde bulunulamaz.
- Bir alt blokta bulunan etikete GOTO ile gönderimde bulunulamaz.
- Alt programın içinden GOTO ile dallanmaya izin verilmez.
- Aykırı kısımdan diğer bölümlere GOTO ile gönderimde bulunulamaz.

## NULL ifadesi

NULL ifadesi hiçbir işlemin yapılmayacağını gösterir. Programın okunmasında kolaylık sağladı için tercih edilebilir. Örneğin;

```
IF oran > 60 THEN
  Komisyon := komisyon * 2;
ELSE
  NULL;
END IF;
```

Bu örnekte oran 60 dan büyükse komisyonun iki katının alınacağına ama değilse hiçbir işlemin yapılmayacağı gösteriliyor. NULL ile tersi bir durumun etkisiz olduğu vurgulanmış olur.

### 4.1.5 İmleçler

Oracle çalışma alanlarını SQL ifadelerini icra etmek ve işlene bilgileri saklamak için kullanır. PL/SQL imleçlerle (cursors) bu çalışma alanlarını kullanmayı ve saklanan bilgilere ulaşmayı sağlar. Diğer bir değişle kayıtların hafızaya getirilme işlemine imleç açma denir. Tek tek kayıtlar bu alanlara getirilir, işlenir ve veritabanına geri gönderilir. İki tip imleç vardır;

1) Kapalı İmleçler: Programlarda kullanılan DML komutları (SELECT, UPDATE, DELETE) için veritabanın otomatik olarak açtığı imleçlerdir. Bölüm 7'deki kayıt_getir prosedüründe kapalı imlece örnek verilmiştir.

2) Açık İmleçler: Kullanıcı tarafından belirli bir amaç için açılan imleç türüdür.

#### 4.1.5.1 Açık imleçler

Sorgulama ile dönecek satır sayısı kriterlere bağlı olarak değişir. Sorgu birden çok satır döndürüyorsa bir imleç tanımlanarak, satırlar işlem görebilirler. Tanımlama işlemi PL/SQL blokların , alt programların ve paketlerin tanımlama kısımlarında yapılır. imleçler üç komutla kontrol altına alınırlar. Bölüm 7'de form örneğinde kayıt_getir prosedüründe imleçlerle yer almaktadır.

1) OPEN

2) FETCH

3) CLOSE

İmleç yapısı aşağıdaki gibi olabilir;

```
DECLARE
 CURSOR <imleç_ismi> IS
  SQL ifadesi
kayıt_tipi_değişkeni <imleç_ismi>%ROWTYPE;
BEGIN
 OPEN <imleç_ismi>;
 LOOP
     FETCH <imleç_ismi> INTO kayıt_tipi_değişkeni;
     EXIT WHEN <imleç_ismi>%NOTFOUND;
          komutlar
         .....
```

```
 END LOOP;
 CLOSE <imleç_ismi>;
END;
```

### İmlecin tanımlanması

Tanımlama bölümünde aşağıdaki yapıya uygun olarak imleçler tanımlanır.

```
CURSOR  <imleç_ismi> [(parametre[, parametre] ...)]
 [RETURN dönüş_tipi] IS select_ifadesi;
```

İmleçler parametre alabilirler. Parametre yazılımı şöyledir;

imleç_parametre_ismi [IN] veri_tipi [{:= | DEFAULT } açıklama]

Örneğin;

```
DECLARE
 CURSOR c1 (parametre1 INTEGER DEFAULT 0,
            Parametre2 INTEGER DEFAULT 100) IS SELECT ....
```

### İmlecin Açılması

İmleçler OPEN komutu ile açılırlar. Henüz satırlar dönmemiştir ama imleç hafızada bir yer kaplarlar. OPEN komutu eğer UPDATE işlemi için imleçi açmışsa ilgili satırları kilit altında tutar. Yazılımı ;

```
OPEN imleç_ismi
```

Eğer imleç parametre alıyorsa;

```
OPEN imleç_ismi(parametre1,parametre2,...) dir.
```

### İmlecin Satırları Getirmesi

FETCH komutu ile SQL ifadesinin sorgulanmasının sonucunda dönen kayıtların yüklenmesi sağlanır. Her kayıt hafızaya geldikten sonra imleç bir sonraki kayda konumlanır ve imleçteki kayıtlar bitene kadar ve bir hata ya da aykırı bir durumla karşılaşılmadığı sürece bu durum devam eder.

```
LOOP
```

```
FETCH c1 INTO kayıt;
EXIT WHEN c1%NOTFOUND;
---işlemeler
END LOOP;
```

**İmlecin Kapatılması**

Açılan her imleç kapatılmak zorundadır. Kapatılan imleçler tanımsız hale dönüşürler. Kapana imleçler tekrar açılabilir.

```
CLOSE c1;
```

### 4.1.6 İstisna kullanımı

PL/SQL de hata ve uyarı durumları istisna ya da aykırı durum olarak adlandırılır. İstisnalar sistemde tanımlıdır ya da kullanıcı tanımlı olabilir. Bazı genel istisnalar sistemde önceden tanımlanmıştır. Örneğin; INVALID_NUMBER ya da TOO_MANY_ROWS.

#### 4.1.6.1 Sistemde tanımlı istisnalar

Oracle tarafından tanımlanmış istisnalar PL/SQL bloklarında EXCEPTION kısmında kullanılırlar. Yazılım dizisi şöyledir;

```
EXCEPTION
 WHEN <tanımlı_istisna_durumu> THEN
    komutlar
```

İstisna durumlarına bazı örnekler;

- CURSOR_ALREADY_OPEN: Açık olan biri imlecin tekrar açılması durumundaki aykırılıktır.
- DUP_VAL_ON_INDEX: Unique indeks yapısını bozan çift kayıt girilmesi esnasındaki aykırı durumdur.
- INVALID_CURSOR: Geçersiz bir imleç işlemi yürütüldüğünde bu aykırı duruma düşer. Örneğin açık olmayan bir imlecin kapatılması.
- INVALID_NUMBER: Karakter string geçersiz bir sayısı gösterirse bunun sayıya dönüşümü aykırı bir durumu gösterir.
- LOGIN_DENIED: Program geçersiz kullanıcı adı ve şifreyle Oracle'a bağlanırsa bu istisna durumu ortaya çıkar.
- NO_DATA_FOUND: SELECT INTO ifadesi satır döndürmezse aykırı duruma düşer.

içersinde RAISE komutu ile çağrılması gerekmektedir.

Örnekte komisyonun 500000000 TL den düşük olması aykırı bir durum olarak IF ifadesinde RAISE ile belirtilmiştir. EXCEPTION bölümünde RAISE olduğunda mesaj verilmesi sağlanmıştır.

```
DECLARE
.....
BEGIN
 IF komisyon < 500000000 THEN
   RAISE az_komisyon;
....
EXCEPTION
WHEN az_komisyon THEN
  MESSAGE('Komisyon 500000000 den düşüktür.');
END ;
```

## 4.2 Tetikler

Tetik, veritabanında belirli tip işlem ve operasyonlar sırasında icra gören prosedürlere verilen isimdir. PL/SQL ve Java kodlarıyla yazılabilen tetikler aynı zamanda C prosedürlerini çağırabilirler (Oracle, 1999b; s20-2).

Tetikler belirli durumlar çalışırlar. DML ifadelerinde yani bir tablonun INSERT, UPDATE , DELETE komutlarıyla modifikasyonunda, DDL ifadelerinde icra edilirken tetikler çalışabilir. Ayrıca veritabanının açılışı, kapanması, hata mesajları gibi sistem olaylarına ve kullanıcının

veritabanına bağlanması ve çıkması gibi kullanıcı kaynaklı olaylara tetikler bağlanabilir.

Tetikler veritabanının özelleştirilmesinde çok faydalıdır ama lüzumsuz kullanımı veritabanının bakımını zorlaştırmaktadır.

Tetikler bütünlük kısıtlamaları ile aynı tip kısıtlamaları oluşturabilirler. Ama yalnızca şu durumlarda tetikler kullanılmalıdır;

- Ayrı veritabanında olan parent ve child tablolara referans kısıtlaması eklemek.
- Bütünlük kısıtlamalarıyla tanımlanamayacak kompleks kurallar oluşturulurken.

### 4.2.1 Tetiklerin bölümleri

Tetiklerin üç temel bölümü vardır;

1) Tetiğin, hangi olay ya da ifadenin tetiklenmesiyle harekete geçeceğini gösteren kısımdır.

Örneğin; UPDATE OF envanter_table ON talep

2) Kısıtlama kısmı.

WHEN (talep.miktar < stok.miktar)

3) Tetik hareketi kısmı PL/SQL blokları, java programlarında oluşur.

### 4.2.2 Tetik tipleri

Tetikleri dört grupta toplayabiliriz;

1) Satır ve ifade Tetikleri

2) BEFORE ve AFTER Tetikleri

3) INSTEAD-OF Tetikleri

4) Sistem ve kullanıcı olaylarında Tetikler

### 4.2.3 Satır ve ifade tetikleri

#### 4.2.3.1 Satır tetikleri

Tablonun satırının tetikte belirtilen durumlarında çalışan tetik tipidir. Örneğin bir UPDATE işlemi aynı anda bir çok satırı işleyebilir, eğer satırları UPDATE esnasında gerçekleşecek tetikleri varsa her satır için tetik çalışır.

#### 4.2.3.2 İfade tetikleri

Bu tip tetikler bir önceki tipin aksine satır sayısına bakmaksızın işleme neden olan ifadeyi dikkate alırlar. Örneğin eğer bir DELETE ifadesi aynı anda birçok satırı silerse, tabloda bir DELETE tetiği varsa yalnızca bir kere çalışır. Satırların değerlerinin önemli olamadığı durumlar için ifade tetikleri satır tetiklerine göre faydalıdır.

### 4.2.4 Önce ve sonra tetikleri

Before ve after tetikleri satr ve ifade tetiklerine uygulanabilirler. DML ifaleriyle çalışan tetikler yalnızca tablolar üzerine yazılabilir. Before tetiği ifadenin icrasından önce çalışan tetiklerdir. After tetiği ise ifadenin icrasından sonra çalışır.

Satır ve ifade tetikleri ile after ve before tetikleri kombine bir şekilde kullanılabilir.

- BEFORE statement trigger
- BEFORE row trigger
- AFTER statement trigger
- AFTER row trigger

# 5. ORACLE YARDIMCI ARAÇLARI

## 5.1 SQL*Plus

SQL*Plus veritabanı dili SQL'in ve onun prosedurel dili olan PL/SQL'in kullanılabildiği bir araçtır. SQL Oracle veritabanına verileri saklamaya ve erişmeye olanak tanır. PL/SQL'de birçok SQL komutunu prosedurel dil mantığıyla kullanmayı sağlar.

SQL*Plus, SQL komutlarını ve PL/SQL bloklarının çalıştırılmasına imkan tanır. SQL/Plus ile şu işlemler yapılabilir (Oracle, 1999e);

- SQL komutları ve SQL*Plus blokları girişi, yazılımı, saklanması, alınması ve çalıştırılması,
- Sorgu sonuçları bir rapor formatında basılabilir, saklanabilir, hesaplamalar yapılabilir ve formatlanabilir,
- Tabloların kolon tanımlamaları listelenir,
- Veritabanındaki verilere erişilir ve kopyalanabilir,
- Son kullanıcıya mesaj gönderilebilir ve cevapları alınabilir,
- Veritabanı yönetimi gerçekleştirilebilir.

SQL*Plus'ı çalıştırabilmek için gerekenler:

Değişik konfigürasyonlarda çalışabilirler. Oracle ve SQL*Plus yüklü olmak zorundadır. Üzerinde çalıştığımız veritabanı Oracle8i ve SQL*Plus 8.1.5 dir. Bir kullanıcı ismi ve şifre ile veritabanına bağlanılır. Kullanıcı ismi ile veritabanında yetkilendirilme sağlanır. Kullanılan işletim sisteminin promtundan SQLPLUS komutu ile SQL*Plus aracı başlatılır. Kullanıcı adı ve şifreyle veritabanına bağlanılır. Bir diğer yol da

SQLPLUS STK/STK ile kullanıcı adı ve şifre bir komut satırında verilebilir.

SQL*Plus aracından çıkış şu şekilde sağlanır.

SQL> EXIT

### 5.1.1 SQL*Plus temel kavramları

- Blok : SQL ve SQL/Plus komutlarının bir grubudur.

- Tablo : Oracle'ın temel depolama ünitesidir.

- Sorgu: SQL'in Select komutuyla bir veya daha fazla tablodan bilgi alınmasına denir.

- Sorgu sonucu: Sorguyla alınan verilerdir.

- Rapor: Sorgu sonuçları SQL*Plus komutlarıyla da formatlanabilir.

### 5.1.2 SQL*Plus temelleri

Üç tip komut SQL*Plus promtundan girilebilir,

1) SQL komutları, veritabanındaki verilerle çalışmak için,

2) PL/SQL blokları, yine veritabanındaki verilerle çalışmak için,

3) SQL*Plus komutları, bunlar sorgu sonuçlarını formatlamak için, seçimleri değiştirmek, SQL komutlarını ve PL/SQL bloklarını saklamak, değiştirmek için.

(;) komutun sonu olduğunu belirtir. Enter'a basılmasıyla SQL*Plus komutu işler ve sonucu ekranda gösterir. Tekrar Enter'a basılmasıyla en son işlene komut tekrar çalışır.

Sql komutları istenilen noktalardan ayrılabilir.

SQL komutunu sonlandırma üç yolla yapılabilir.

- (;) ile,
- satırda yalnız bir (/) ile,
- boş bir satır ile.

#### 5.1.2.1 SQL ifadelerinin kullanımı

SQL*Plus edit komutları Çizelge 5.1'de yer almaktadır.

Çizelge 5.1 SQL*Plus edit komutları

| KOMUTLAR | KISALTMASI | AÇIKLAMA |
|---|---|---|
| APPEND | A text | Satır en sonuna text eklenir. |
| CHANGE | C /eski/yeni | Eski yeni ile değiştirilir. |
| CHANGE | C /text | Text satırdan silinir. |
| CLEAR BUFFER | CL BUFF | Tüm satırları siler. |
| DEL | | O anki satırı siler. |
| DEL n | | n. satırı siler. |
| DEL * | | O anki satırı siler. |
| DEL LAST | | Son satır siler. |
| DEL m n | | m ile n arasındaki satırları siler. |
| DEL * n | | O ankş satırdan n kadar olan satırları siler. |
| INPUT | I | Bir veya daha fazla satır ekler. |
| Input text | I text | Text'i içeren bir satır ilave eder. |
| LIST | L | SQL buffer daki tüm satırları listeler. |
| LIST n | L n | Satır n'yi listeler. |
| LIST * | L * | O anki satırı listeler. |
| LIST n * | L n * | Satır n den geçerli satıra kadar listeler. |
| LIST LAST | L LAST | Son satırı listeler. |
| LIST m n | L m n | m ve n arasındaki satırları listeler. |
| LIST * n | L * n | Geçerli satırdan satur n ye kadar listeler. |

**Komutların komut dosyasında saklanması**

SQL ifadesi, PL/SQL bloğu ya da SQL*Plus komutları sonraki kullanımlar için saklanabilir.

Bu ifadeler komut dosyasında saklanır.

SAVE dosya_adı [REPLACE | APPEND]

Replace: varolan dosyanın yerine kaydeder.

Append: dosyanın sonuna ekler.

Üç şekilde komut dosyası yaratılabilir.

1) Bir komut girilir ve buffer daki içerik kaydedilir.

SQL> SAVE dosya_adi

Oluşan dosyanın uzantısı SQL olarak eklenir. İstenirse kaydedilirken verilen isme uzantısı da eklenir.

2) INPUT ile komutlar girilir ve tampondaki içerik kaydedilir. İlk olarak SQL*Plus komutları girilir ardından SQL komutları ya da PL/SQL blokları girilir.

Buffer temizlenir.

```
SQL> CLEAR BUFFER

SQL> INPUT
  1  COLUMN MAAS HEADIND KOMISYON FORMAT 999,999,999
  2  SELECT  DEPT_NO,PERSONEL_NO,MAAS
  3  FROM MAAS_TABLE
  4  WHERE MESLEK = 'MEMUR'


SQL> SAVE SATIS
Created file SATIS
```

3) EDIT kullanarak host sistem text editorunden dosya yaratma. Çünkü SQL*Plus komutları buffer da saklanmaz.

**Diğer SQL*Plus komutları**

- GET <dosya adı>: İşletim sistemine kaydedilen komut dosyalarının tekrar hafızaya

yüklenmesini sağlar.

SQL> GET 'D:\SQL_PLUS\SATIS'

- START <dosya adı> : İşletim sistemine kaydedilen komut dosyalarının çalışmasını sağlar.

SQL>START SATIS

Ayrıca @ işaretide START ile aynı işi görür, başına koyulan dosyayı çalıştırır.

SQL>@SATIS

- SPOOL <dosya adı.uzt> OFF | ON : Çıktılar dosyaya yazılır. Uzantısı lst ya da lis'dir. İşletim sistemine kaydedilen bu dosyaların sonradan çıktıları da alınabilir. OFF spool işlemini durdurur, OUT ise işlemi durdurur ve çıktıları sistemin varsayılan yazıcısına gönderir.

**SQL*Plus'daki sistem değişkenlerini ayarlama**

SQL*Plus SET komutuyla sistem değişkenleriyle ilgili ayarlamalar yapılır. Ayarların durumları ON ya da OFF ya da bir sayıya karşılık gelir.

SET <parametre> <off | on | sayı> şeklinde kullanılırlar.

Bazı set komutları:

- ECHO {OFF | ON} : ECHO ON durumunda ise kayıtlı bir komut dosyası çağrıldığında ekranda görülür, eğer OFF durumdaysa ifade ekrandan görülmez.
- HEADING {OFF ON} ON durumunda kolon başlıkları gösterilir, OFF durumda gösterilmez.
- LINESIZE {<sayı>}: Bir satır gösterilebilecek maksimum karakter sayısı belirler. Standart değeri 80'dir.
- NUMFORMAT {999999}: Nümerik verilerin formatını belirler.
- PAGESIZE{<sayı>}: Bir sayfada görünecek maksimum satır sayısını belirler. Standart değeri 24'dür.
- PAUSE {OFF | ON}: ON durumda iken her sayfadan sonra enter tuşuna basılamasını ister. Standart değeri OFF'dur.
- TIMING{OFF | ON}: Sorgu çalıştıktan sonra dönen kayıtların ne kadar sürede geldiğini gösterir. Standart değeri OFF'dur.

Set komutlarının değerini görmek için SHOW <parametre> formatında yazmak yeterlidir.

SQL>SHOW PAGESIZE

Pagesize 24

SHOW ALL komutu ile tüm set komutları değerleriyle beraber görünür.

- COL <kolon adı> FORMAT değer : Verilen kolonun tipine göre formattan sonra yazılan değer değişir. Kolon bir karakter ise değer A15 olabilir.

COL isim1 FORMAT A15

Bu formatın anlamı isim1 kolonunun 15 karakterlik bir alanda listeleneceğidir.

COL maas FORMAT 999.999.999.999.999

Nümerik veri tipli kolonlarda ise yukarıdaki örnekteki gibi verilir.

**SQL*Plus'da değişken tanımlama**

- ACCEPT :

ACCEPT değişken [NUMBER|CHAR] [PROMPT TEXT | NOPROMPT] [HIDE]

NUMBER|CHAR : değişkenin tipini belirler.

NOPROMPT : Prompt göstermeden bir satır aşağıdan acept bekler.

HIDE : Girilen değeri göstermez.

SQL>ACCEPT maas NUMBER PROMPT 'Komisyon'

Komisyon : 300000

SQL>ACCEPT sifre CHAR PROMPT 'Sifre:' HIDE

Sifre:

SQL>ACCEPT FIYAT NUMBER NOPROMPT

1000000

- DEFINE: Değişkenler tanımlanır ve değişkene karakter değerler atılabilir.

DEFINE [değişken] | [değişken=text]

Define REM = 'MAAS*12+nvl(KOMISYON,0)

select isim1,meslek,&rem

```
from personel
where &rem > 1000;
```

şeklinde kullanılabilir.

- UNDEFINE değişken : Tanımlı olan değişkenlerin tanımlarını ortadan kaldırır.

&degisken : Komut dosyası her çalıştığında kullanıcıdan bir değer bekler.

```
select isim1,meslek,maas
from personel
where meslek > '&meslek_giriniz';
```

çalıştırıldığında;

```
Enter value for meslek_giriniz : MEMUR
```

& sayesinde değişken dışardan alınır ve listelenir.

&&degisken : : Komut dosyası ilk çalıştığında kullanıcıdan bir değer girmesini sağlar, daha sonraki çalışmalarında sormaz.

## 5.2 Oracle Developer 6i

Oracle Developer 6i farklı platformlarda uygulama tasarımı yaratılmasını ve çalıştırılmasını sağlayan bir grup programlama aracından oluşur. Proje, form, rapor, grafik, sorgu ve program unit geliştiricileri içen Oracle Developer 6i, tüm bu araçlarla entegre olarak çalışır. Bu sistem kod yazmadan veritabanı tanımlamalarından uygulama geliştirmeyi sağlar.

Her uygulama için çoklu veritabanı bağlantısıyla lokal, istemci/sunucu ve web desteğie de sağlamaktadır.

En önemli ve en çok kullanılan araçlarform ve rapor gelişticilerdir. Form Builder ve Report Builder'ın ortak bileşenleri şunlardır;

- Object Navigator
- Property Palette
- Layout Editor
- PL/SQL Development Environment

- Object Navigator tüm nesneleri hiyerarşik yapısı ile gösterir. Bu yapı üzerinden bir nesne

seçilebilir, edit edilebilir, silinebilir ya da isminden aratılabilir.

- Nesnelerin özelliklerini gösteren, değişikliği imkan tanıyan grafik tabanlı yapılardır.

- Uygulamaların ara yüzünün tasarlandığı ekrandır.

- Form Builder, PL/SQL dilinin kullanımına imkan tanığı gibi geliştirme safhasında da bu araç ile kolaylık sağlar.

### 5.2.1 Form geliştirici

Form Builder'ın birçok modülü vardır. Bunlar;

- Form Modülü: Kullanıcıyla birebir etkileşime giren veri alanlarını içerir.

- Menü Modülü : Form modüle eklenebilen yararlı bir modüldür. Her form modülün bir varsayılan menü modülü vardır. sonrasında bu modül özelleştirilebilir.

- PL/SQL kütüphanesi : Diğer modüller kullanılacak program ünitelerinin kütüphanesidir.

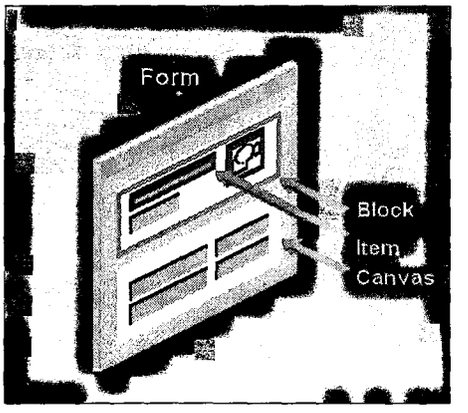
- Nesne kütüphanesi : PL/SQL kütüphanesi gibi diğer modüller tarafından kullanılır. Tekrar kullanılacak nesnelerin gruplanıp saklanmasını sağlar. Diğer formalarda kopyalama ya da alt sınıf mantığıyla kullanılabilir. Bu durum formun nesneye dayalı yapıyı desteklemesine bir işarettir. Alt-sınıf (subclass) edilen nesneler aralarındaki baplantıları isdikleri sürece korurlar.

- Formun temel bileşenleri aşağıdadır.

1) Blok

2) Alan

3) Kanvas




Şekil 5.1 Form temel bileşenleri

Alanlar formun ilk bileşenleridir. Kullanıcıya değerleri gösteren ara yüz nesneleridir. Bu yapılar sayesinde kullanıcılarla etkileşim sağlanır. Birçok tip alan bulunmaktadır. Bazıları, text, button, check box, radio, display item, list item, sound item dır. Alanlar mantıksal olarak bloklar tarafından kanvasın üzerinde gruplanırlar.

Bloklar her formda en az bir tane olmak zorunda olan, alanların mantıksal sahipleridir. Her alan bir bloğa aittir.

Bir bloktaki alanlar mantıksal olarak birbiriyle ilişkilidir. Aynı tablonun alanları bir blokta yer alarak görüntülenmesi, saklanmaları ve manipülasyonları bloklar sayesinde bir mekanizma altında toplanır.

Kanvaslar alanların ve grafiklerin sıralandıkları görsel nesne yüzleridir. Bir kanvasta bir veya daha fazla bloğa ait birçok alan gösterilebilir. Tipleri;

- Content
- Tab
- Vertical
- Horizantal
- Stacked

Kanvas ve alanlarının gösterilmesi için pencerelere ihtiyaç vardır. Bir kanvası bir resim gibi düşünürsek pencereyi de kanvasın çerçevesi olarak hayal edebiliriz.

Alanlar her bir blok içinde , her bir bloğunda form içinde bir sırası vardır. Eğer diğer alana geçiş komutu verilirse, sıradan kendisinden sonraki alana geçer. Eğer alan başka bir kanvasta ise form builder otomatik olarak kanvası gösterir.

İki tip blok vardır. Biri veri bloğu diğeri kontrol bloğudur. Ver bloğu veritabanındaki bir tabloyla ya da görüntüyle, gömülü prosedürlerle birbirlerine bağlıdır. Kontrol blokları veritabanı ile bağlantılı değildir, işlemlerini belirtirler.

Veri blokları arasında ana anahtar ve yabancı anahtar ile birbirine bağlanabilir. Form builder üst ve alt ilişkisini kuran otomatik PL/SQL kodları üretebilir ya da editörden kodlanabilir.

**Tetikler**

Formda en sık kullanılan nesnelerden biri tetiklerdir. Bu nesneler bir olay yani herhangi bir hareket üzerine çalışan PL/SQL bloklarıdır. Tetikler form, veri bloğu ya da alan bazında olabilir. Tipleri;

- Anahtar tetikleri : Fonksiyonu yerine getiren tuşa basıldığında aşağıdaki tetikler çalışır. Örneğin; KEY-NEXT-ITEM tetiği bir alan üzerinden bir sonrakine geçiş için sonraki alan tuşuna basıldığında çalışır. Bunun üzerine alan kontrolleri yazılabilir.

  - KEY-NEXT-ITEM :Bölüm 7'de Şekil 7.17'de bir örneği vardır.
  - KEY-COMMIT
  - KEY-EXEQRY

  .....

- Olaya bağlı tetikler : Formda gerçekleşen her hareketi bir olay tipine dahil edebiliriz. Bu olayları oluş zamanlarına tetikler sınıflandırılmıştır. Örneğin; ON-COMMIT tetiği ile onaylama işlemi sırasında bir mesaj verilip, kullanıcı bilgilendirilebilir.

  - PRE tetikleri
    - PRE-BLOCK
    - PRE -CHANGE
    - PRE-COMMIT
    - PRE-DELETE
    - PRE-UPDATE
  - POST tetikleri
    - POST-BLOCK
    - POST-CHANGE
    - POST-COMMIT
    - POST-DELETE
    - POST-UPDATE
  - ON tetikleri
    - ON-CLEAR-BLOCK
    - ON-DELETE

- ON-ERROR
- ON-INSERT

- WHEN tetikleri
  - WHEN-NEW-FORM-INSTANCE
  - WHEN-NEW-BLOCK-INSTANCE : Bölüm 7'de şekil 7.21'de bir örneği yer almıştır.
  - WHEN-NEW-RECORD-INSTANCE
  - WHEN-BUTTON-PRESSED : Bölüm 7'de şekil 7.18'de ve 7.22'de birer örneği yer almıştır.
  - WHEN-CHECKBOX-CHANGED

INSERT, UPDATE,DELETE DML komutlarını POST,PRE ve ON tetikleri vardır. blok veya kayıt bazında tanımlanabilirler. COMMIT ise form bazında PRE ve POST tetikleriyle kontrol altına alınabilir.

Diğer bir nesne kullanıcı tanımlı program üniteleridir. Formda bir veya daha fazla program ünitesi bulunabilir. Bölüm 72'de Şekil 7.23'de program ünitesinde yer alan bir prosedüre örnek verilmiştir. Bu yapılar PL/SQL bloklarından oluşan prosedürler, fonksiyonlar ve paketler olabilir.

## 5.3 Rapor geliştirici

Raporlama bilgilerin kullanıcılara ulaşmasına yardımcı olan karar destek araçlarından biridir. Elektronik ortamda verinin depolanması kadar eskidir. Bu kullanıcılara farklı araçlar farklı imkanlar sunarlar. Sorguların entegrasyonunu, analizini sağlarlar ve raporlarlar.

Report Builder rapor geliştirme aracı olarak kullanılmaktadır. Report Builder şunları içerir;

- SQL ifadelerin grafik sunumlarıyla hazırlanması,
- Rapor dizaynı prosesini yardımcı ekranlarla yürütme,
- Canlı ön izleme ekranında edit imkanı,
- Grafik desteği,
- Raporların nasıl çalışacağını özelleştirmek için kodlama olanağı,
- Web ile uyumlu raporlar hazırlama,
- Çıktı fotmatları HTML, HTMLCSS, XML, PDF, PCL, ASCII olabilir,
- Çalışır durumda özelleştirme olanağı,
- İstemci tarafı parametrelerini doğrulamada JavaScript kullanılabilir,
- PL/SQL prosedürlerin içinden dinamik SQL ifadelerini icra edebilme,
- Nesne oryantasyonunu dstekler.

### 5.3.1 Rapor geliştiricisi bileşenleri

Form builder'dan farklı olan bileşenler aşağıdadır (Oracle, 1999c);

Live Preview

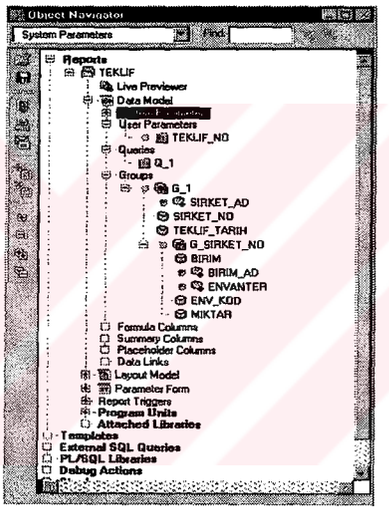
Üretilen raporu izlemeye olanak tanır . Aynı zamanda görüntüsünde değişiklik yapılmasına da izin verir.

Data Model

Rapor için data model nesneleri bu bölümde tanıtılır. Aralarındaki ilişkiler grafiksel olarak belirtilir.

Layout Model

Raporun tasarımının oluşturulduğu bölümdür.




Şekil 5. 2 Nesne erişimcisi (Oracle Forms Builder 6i)

#### 5.3.1.1 Veri modeli

Raporda kullanılacak verinin yapısının tanımlandığı bölümdür. Aşağıdaki bileşenlerden oluşur.

- Sorgular
  - Gruplar
  - Kolonlar
- Veri bağlantıları

- Parametreler

Sorgular verilerin SELECT ile seçildiği yapılardır.

Gruplar sorgulanan verinin sınıflandırılmasını organize eder. Veriler sorgunun her kayıt getirmesinde gruplanırlar. Bölüm 7'de rapor örneğinde şeikl 7.10'da bir data model yer almaktadır.

Kolonlar genelde sorgudan getirilen kolonlardır. Ayrıca şu tiplerde kolonlar vardır;

- Formula :Bölüm 7'de şekil 7.11'de bir örnek verilmiştir.
- Placeholder
- Summary

Veri bağlantıları sorguların birleştirilmesini sağlar.

Parametreler sistem parametreleri ve kullanıcı parametreleridir.

#### 5.3.1.2 Arayüz modelleme nesneleri

- Frame
- Repeating frame
- Alanlar
- Boilerplate

Her bir repaeting frameler bir gruba bağlıdır. Repeating framelerin içini alanlar ve boilerplate yapıları oluşturur. Her bir kayıt için tekrarlanırlar. Bağlı oldukları grubun dışındaki yalnızca üst grupların kolonlarını içerebilirler.

Fremeler yalnızca bir kere basılırlar. Diğer nesnelerin tümünü içerirler.

Alanlar verileri ve diğer değerleri, boilerrplate ise metin ve grafik içerir.

#### 5.3.1.3 Parametre form nesneleri

Alanlar ve boilerplate'lerden oluşur. Çalışma esnasında kullanıcıdan ya da sistemden sağlanan değerlerdir. Bölüm 7'de şekil 7.12'de bir parametre ekranı yer almaktadır.

#### 5.3.1.4 Rapor dosyalarını depolanması

Report builder modülü raporları işletim sisteminde sakladığı gibi report builder veritabanı tablolarında da saklar. Genelde sistem dosyası olarak saklanan rapor tipleri şunlarıdır;

- .rdf :Kaynak kodu ve komutlarını içeren tanımlama dosyasıdır. Çalıştırılabilir ve report

builder'da modifiye edilebilir.

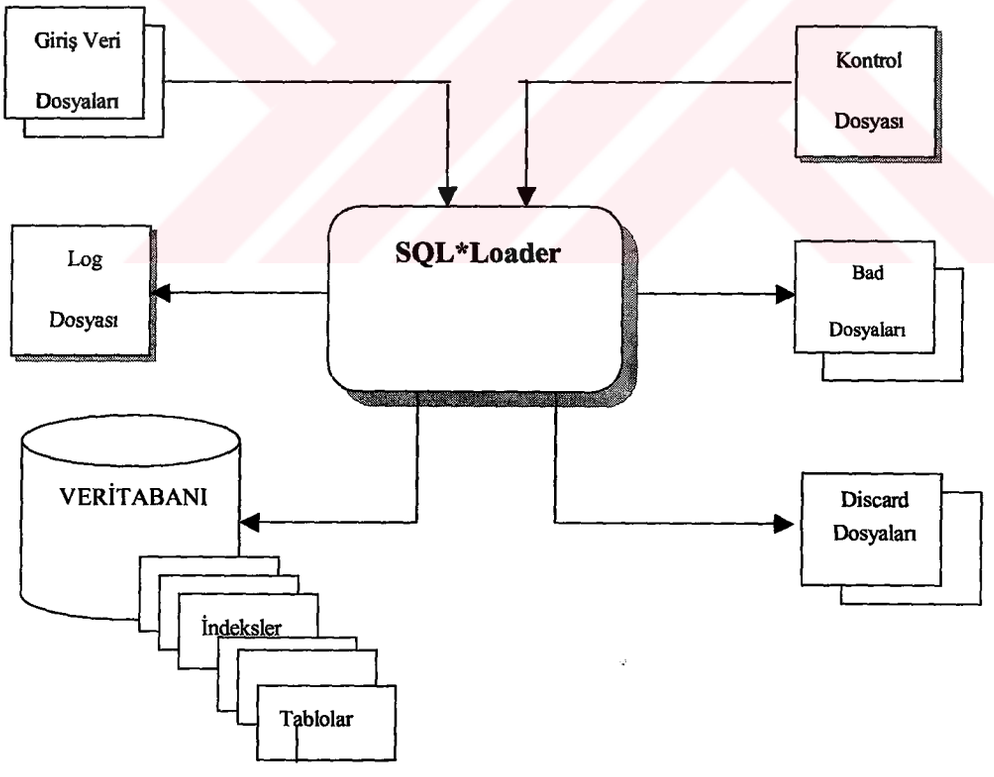
- .rep :Çalışabilen bir dostadır, kaynak kodu ve komutlarını içermez. Çalıştırılabilir ama report builder'da modifiye edilemez.
- .rex :Kaynak kodu ve komutlarını içeren tanımlama dosyasıdır. Çalıştırılamaz yanlızca kodu kontrol eder ve debug işleminde kullanılır.

# 6. VERİTABANINA VERİNİN YÜKLENMESİ VE ÇIKARTILMASI

## 6.1 Sql*Loader Temelleri

SQL*Loader veritabanının dışındaki ASCII formattaki dosyalardaki verilerin Oracle veritabanın içindeki tablolara yüklenmesini sağlayan bir araçtır(Oracle,1999g). Özellikleri;

- Veri dosyalarındaki veri formatları üzerine kısıtlamalar koyabilir,
- Aynı yükleme oturumuyla veriyi çoklu tablolardan yükler,
- Koşullu olarak veri yükler,
- SQL fonksiyonları kullanılarak, veri yüklemeden önce işlenebilir,
- Belirli kolonlarda, tek sıralı anahtar geliştirebilir,
- İşletim sisteminin dosya sistemini kullanarak veri dosyalarına erişebilir,
- Disk ya da teyp'den veri yükleyebilir,
- Hataların raporlanması sayesinde oluşan kesilmeler giderilir,
- DB2, Loader ile uyumludur. SQL*Loader da kullanılan kontrol dosyaları DB2'da da kullanılabilir.




Şekil 6.1 SQL*Loader'a genel bakış

Şekil 6.1'de gösterildiği gibi veritabanına yüklemede girdi olarak veri dosyası ve kontrol

dosyası alınır. Tablolara yüklenecek veriler veri dosyasında yer almaktadır. Bu verilerin adresi, yükleme modu, koşulları ve yükleneceği tablo bilgileri kontrol dosyaların bulunur. Yükleme başladıktan sonra yapılan işlemler log dosyasında tutulur. Tabloya yüklenirken sorun çıkaran kayıtlar bad dosyalarına atılır. Kontrol dosyasındaki şartlara uymayan veriler ise discard dosyasına atılmaktadır. Dolayısıyla yükleme işleminin beş tip dosya kullanılmaktadır;

1) Veri Dosyaları

2) Kontrol Dosyaları

3) Log dosyaları

4) Bad Dosyaları

5) Discard Dosyaları

### 6.1.1 Kontrol dosyaları

Kontrol dosyaları, SQL*Loader'ın öngördüğü formatta yazılmış metin dosyalarıdır. SQL*Loader'ın yerine getireceği görevi tanımlar. SQL*Loader'ın verileri nerede alacağı, nasıl bölüp ayıracağı ve nereye kaydeceği gibi konulara açıklık getirir.

Kesin sınırlarla tanımlanmamasına rağmen, bir kontrol dosyasının üç bölümden oluştuğu söylenebilir.

1) Session-wide bilgiler

- Genel seçenekler (örneğin; satırlar, atlanacak satır sayısı gibi)
- INFILE ile okunacak verinin yeri
- Veri karakter kümesi tanımı

2) Bu bölüm bir ya da daha fazla sayıda "INTO TABLE" bloğu içerir. Her bir blok, verinin yükleneceği tablo adını ve bu tablonun kolon bilgilerini göstermektedir.

3) Bu bölüm seçimliktir, eğer varsa yüklenecek verileri içerir.

Kontrol dosyasının yazılımında şu özellikler vardır;

- Söz dizimi serbest formattadır ve satırlarda dağınık olarak yazılabilir,
- Büyük küçük harfe karşı duyarsızdır ama stringler tek ya da çift tırnak içinde yazılmalıdır

ve küçük büyük harf farklı algılanır,

- Açıklamalar "– " ile başlayan satırlara yazılabilir,

SQL*Loader'da bazı kelimelerin kesin bazı anlamları vardır eğer bunlar kontrol dosyasında başka bir görevde kullanılmak isteniyorsa tek ya da çift tırnak içinde yazılmalıdır.

### 6.1.2 Giriş verileri ve veri dosyaları

SQL*Loader'a kontrol dosyalarının dışında veriler de giriş yapmaktadır Okunacak veri dosyası ya da dosyaları kontrol dosyasında "INFILE" komutu ile belirtilir. SQL*Loader veri dosyasındaki verileri kayıt olarak görür. Şu tip veri dosyaları bulunmaktadır;

- Sabit Formatlı Kayıt
- Değişken Formatlı Kayıt
- Sürekli Formatlı Kayıt

#### 6.1.2.1 Sabit formatlı kayıtlar

Veri dosyasındaki tüm kayıtlar aynı uzunlukta ise "Dosya sabit formattadır" denir. Bu tip formatın diğerlerine nazaran daha az esnek olmasına rağmen performansı daha iyidir.

INFILE <veridosyası_adı> "fix n"

Bu belirtim ile SQL*Loader veri dosyasındaki kayıtları n byte uzunluğundaki parçalara böler.

Örnek:

```
Load data
infile 'ornek_dat' "fixed 11"
into table 'ornek_2'
fields terminated by "," optionally enclosed by ""
( col1 char(5),
 col2 char(7))

ornek_dat:
001,  rewe, 222,  qweer
0005,  rety,
2,"ffg"
```

Herbir kayıt 11 karakter olarak belirtilmiştir.

### 6.1.2.2 Değişken formatlı kayıtlar

Bir veri dosyası değişken tipde tanımlanmış ise SQL*Loader her bir kaydın başlangıcından itibaren kaç uzunluğunda veri okuyacağını bilmelidir. Bu format sabit formata göre bazı esneklikler getirir ve stream formata göre de performansı daha iyidir.

INFILE "veridosya_adı" "var n"

n: Bir kaydın byte uzunluğudur. Eğer n belirtilmezse varsayılan değeri beştir. Ancak n 2^32 −1 den büyük olamaz.

Örnek:

```
load data
infile 'ornek2.dat' "var 3"
into table ornek_2
fields terminated by ',' optionally enclosed by '"'
(col1 char(5),
col2 char(7))
```

ornek2.dat:

```
009hello,cd,010world,im,
012my,name is,
```

### 6.1.2.3 Sürekli formatlı kayıt

Bu format türü diğerlerine nazaran daha esnek bir yapıdadır. Bununla beraber performansı biraz zorlar. Kayıtların büyüklüğü belirtilmez, SQL* Loader kayıtları "record terminator" lar yardımıyla formatlar.

Yorumlanacak veri dosyasının süreklilik formattaki belirtimi şöyledir;

INFILE <veridosya_adi> ["str 'sonlandirici_string'"]

'sonlandırıcı_string' (terminator-string) bir alfa-sayısal karakter tipindedir. Bununla beraber aşağıdaki durumlarda hexadesimal string olarak belirtilmelir.

- Sonlandırıcı_string özel (çıktısı olmayan/non-printable) karakterler içerirse,
- Sonlandırıcı_string yeni satır ya da aynı satırı tekrar yazdıran karakterler içerirse,
- Belirlenen sonlandırıcı_string, veri dosyası için karakter serisi client'ınkinden farklı ise.

Eğer terminator_string belirtilmemişse, varsayılan yeni-satır karakteri olur. Satır-besleme Unix ortamda, Microsoft ortamda carriage return satır-besleme ile takip edilir.

**Örnek :**Örnekteki sürekli kayıt formatında terminator string bir hex-string'dir. ASCII karakter serisinde X'7C0a' olarak gösterilen karakter '|' işaretinden '\n' yeni-satır karakteriyle çevrilir. Örnekteki veri dosyası iki kaydı içermektedir ve her ikisi de '|\n' ile bitirilmiştir.

```
load data
infile 'ornek3.dat' "str X'7c0a'"
into table ornek3
fields terminated by ',' optionally enclosed by '"'
(col1 char(5),
col2 char(7))
```

Veri dosyası şu kayıtları içermektedir;

ornek3.dat:

```
pelin,senturk,|
tulin,zafer,|
```

### Mantıksal kayıtlar

SQL*Loader giriş verilerini kayıt format belirteçlerine uygun fiziksel kayıtlar olarak organize eder. Varsayılan olarak bir fiziksel kayıt bir mantıksal kayda eşdeğerdir. SQL*Loader birçok fiziksel kaydı bir araya getirerek mantıksal kayıtlar oluşturabilir.

- Sabit sayıdaki fiziksel kayıtları birleştirerek bir mantıksal kayıt olarak şekillendirmek,
- Bazı durumlar oluştuğunda fiziksel kayıtları mantıksal kayıtlara dönüştürmek

### Veri alanları

Bir mantıksal kayıt biçimlendirilirse, üzerinde bulunacak alan düzenlemesi de yapılmış olur. Kontrol dosyasındaki belirteçler sayesinde verinin hangi kısmının hangi alana bağlanacağı belirtidir. İki ya da daha fazla alan belirtecinin aynı veriyi kullanabilmesi mümkündür. Ayrıca bir mantıksal veri, kontrol dosyası alanı ile belirtilmeyen verileri de içerebilir. Çoğu kontrol dosya alan belirteci mantıksal kaydın belirli bir parçasını içerebilir.

- Veri alanının başı veya sonunun byte pozisyonu ya da her ikisi belirtilebilir. Bu belirtim formu fazla esnek olmasa da performansı iyidir.

- Bitimi belirten string'lerin her biri veri alanının pozisyonunu tanımlar. Eğer veri alanının başlangıç pozisyonunun byte belirlenmişse, delimited veri alanlarının son verinin bitiminden başladığı varsayılır.

- Veri alanlarının uzunlukları ya da byte ofsetleri belirtilebilir. Bu yolla her alan bir öncekinin bittiği yerden belirli sayıdaki byte kadar başlar ve bu uzunlukta devam eder.

- Uzunluk-değer veri tipi de kullanılabilir. Bu durumda veri alanının ilk x byte uzunluğu geri kalan veri alanının uzunluğunu gösterir.

#### 6.1.2.4 Veri aktarımı ve veri tipinin belirlenmesi

Şekil 6.2'de veri dosyasındaki veri alanlarının bir geleneksel yolla yüklemesi ile veritabanındaki kolonlara çevirmeyi göstermektedir. Bu yol direk yolla yükleme ile kavramsal olarak benzer fakat uygulamada farklıdır. Diyagramın başında bir ya da fazla veri alanına sahip bir veri dosyası gösteriliyor. Diyagramın alt kısmını da veritabanı kolonuna aktarımda izlenecek yol oluşturmaktadır.

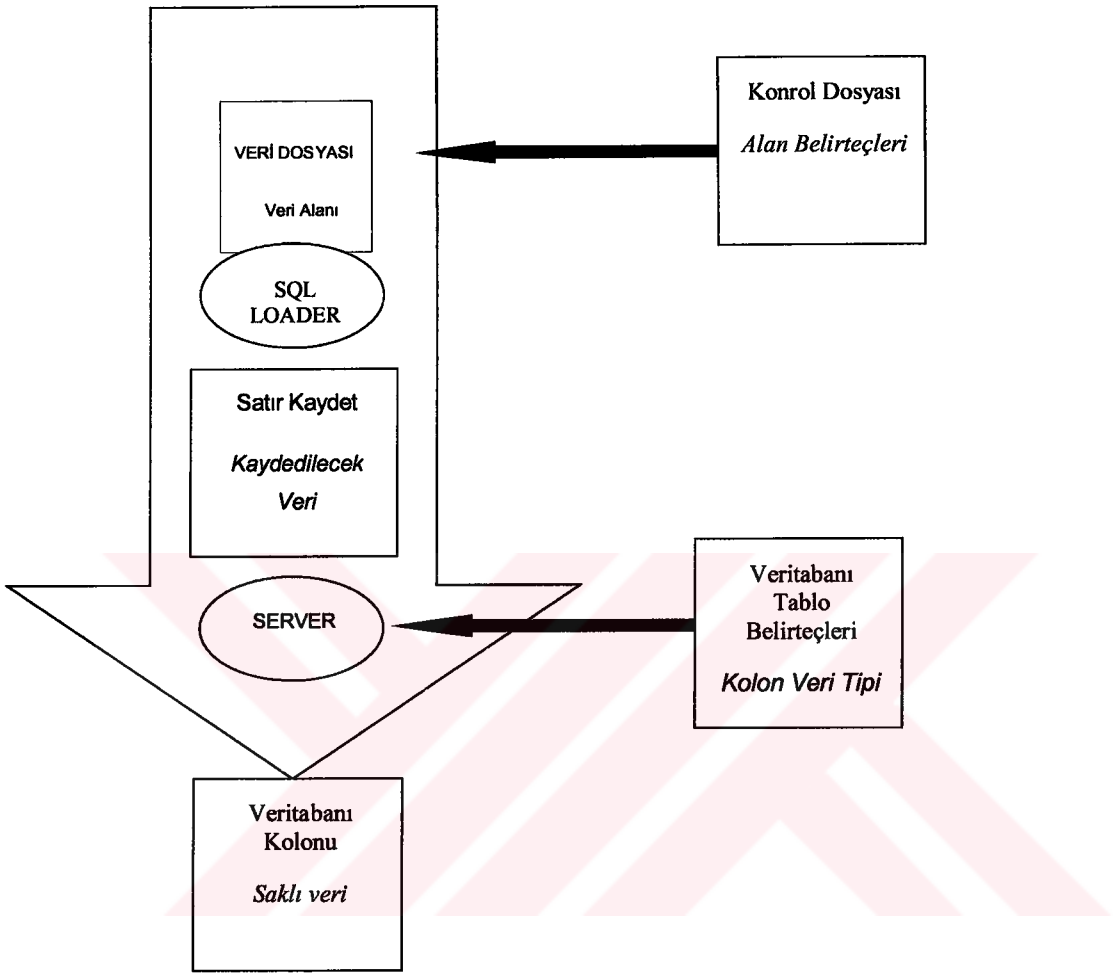
Alan belirtimleri, veri dosyasının formatının SQL*Loader'a nasıl yorumlanacağını aktarır. Oracle Server verileri çevirir, veritabanındaki kolonlara verilen veri tipleri doğrultusunda kaydeder. Buradaki önemli nokta SQL*Loader'da kontrol dosyasında tanımlanan veri tipiyle kolonların veri tipi ile aynı olmayabilir.

Çizelge 6.1 Aktarımdaki veri dönüşümü

| SQL*LOADER | ORACLE DATABASE |
|---|---|
| CHAR | CHAR |
| CHAR | VARCHAR2 |
| INTEGER EXTERNAL | NUMBER |
| DECIMAL EXTERNAL | NUMBER |

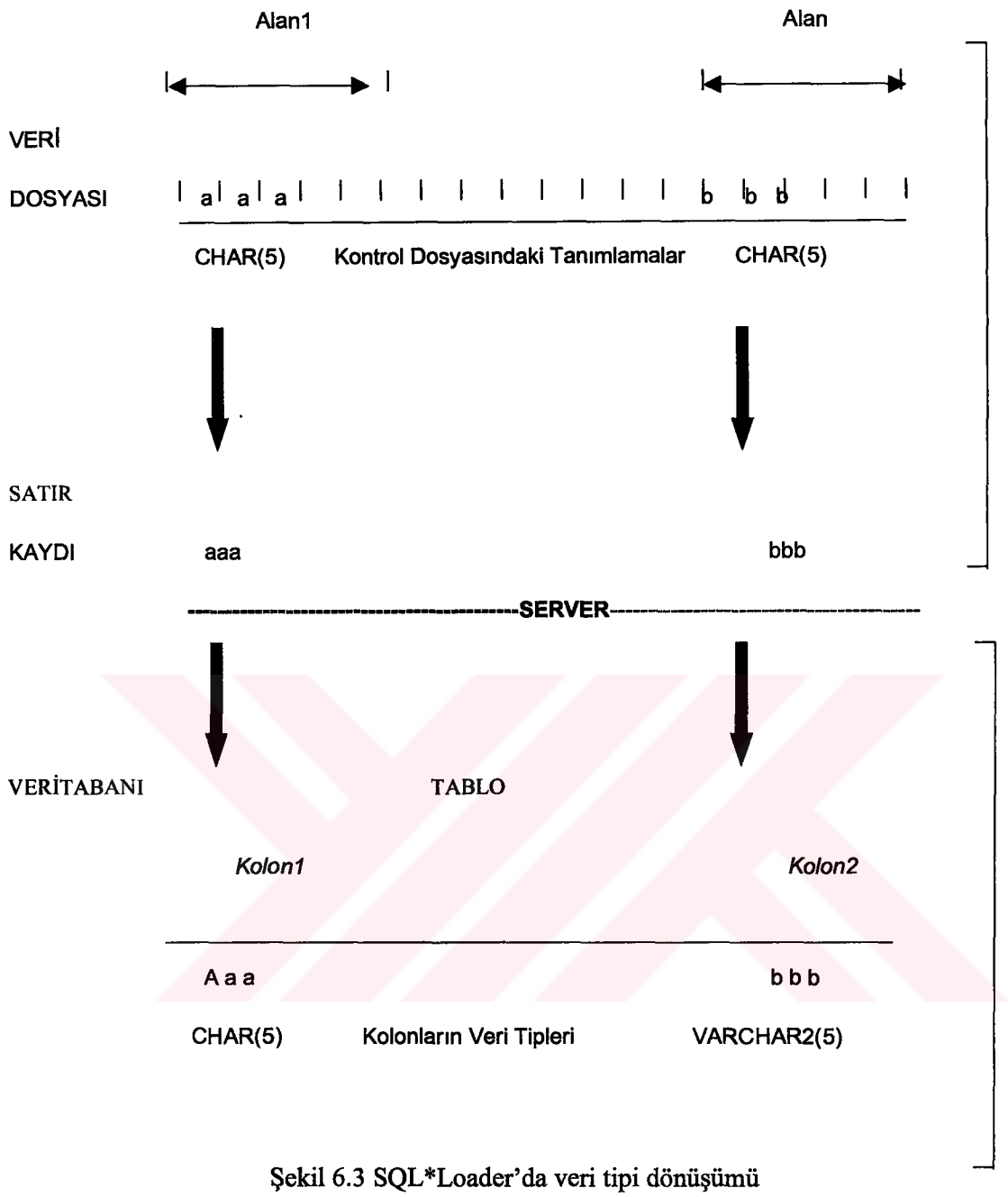
SQL*Loader kontrol dosyasındaki alan tanımlamalarını giriş verilerini sınıflamak için kullanır ve SQL kayıt durumuna benzeyen bir dizi oluşturulur. Kayıt işlemi Oracle sunucusuyla gerçekleşir. Oracle Sunucusu kolonların veri tipini kullanarak verileri tablolara yükler. İki tane çeviri adımı vardır;

1. SQL*Loader veri dosyasındaki bir alanı tanımlar, veriyi yorumlar ve Oracle sunucuya dizi tamponu üzerinden geçirir.
2. Oracle Server veriyi kabul eder ve onu veritabanında saklar.




Şekil 6.2 Giriş veri alanlarının oracle veritabanı kolonlarına transferi

Şekil 6.3'de karakter olarak tanımlanmış iki alan bulunmaktadır. Kontrol dosyasında nasıl tanımlandığı şekilde belirtilmiştir. Burada dikkat edilmesi gereken nokta kontrol dosyasındaki karakter tipi ile veritabanındaki karakter tipini farklı olmasıdır. Hatta bu karakter değil de bir sayı olarak da veritabanına yazılabilirdi. A beş uzunluğunda bir karakter olarak tanımlandı.Buna göre (aaa) verisi sola dayalı olarak kolona yazılır ve kalan uzunluğa da boşluklar atanır.B kolonu ise değişken kolon uzunluğunda tanımlandı, maksimum değeri beş olmak üzere (bbb) veri atanınca bu kolonun uzunluğu üç karakter oldu.




Şekil 6.3 SQL*Loader'da veri tipi dönüşümü

Alanın adı SQL*Loader'a hangi kolona kayıt edeceğini belirtir. Şu noktalar önemlidir;

- Veri alanının adı yüklemenin yapılacağı tablodaki kolon isimlerine bağlıdır.
- Alanların veri tipleri SQL*Loader ın veri tanıması ve aktarması için belirtilir. Önemli olan veri tiplerinin kolonların veri tiplerini etkilememesidir. Yukarıdaki örnekte char'ın varchar2 olması gibi.
- Veri kontrol dosyasında belirtilen tipten veritabanındaki tipe dönüştürülür.

### 6.1.3 Reddedilen kayıt dosyası

Reddedilen kayıt dosyası (bad dosyası) Oracle ya da SQL*Loader tarafında reddedilen kayıtları içerir. SQL*Loader giriş formatı geçersiz olan verileri kabul etmez. Örneğin maksimum uzunluğunu aşan alan SQL*Loader tarafından reddedilir. Bu kayıtlar bad dosyasında yer alır.

SQL*Loader'ın bir kaydı kabulunden sonra Oracle tarafındaki işlemler başlar. Eğr Oracle geçerli bir kayıt olarak yorumlarsa kaydedilir aksi taktirde bu kayıt bad dosyasına atılır. Örneğin key unique olmayabilir ya da null değerindedir ya da Oracle veri tipi için geçersiz olabilir.

### 6.1.4 Elenen kayıt dosyası

Elenen kayıt dosyası (discard dosyası) eğer kontrol dosyasında belirtilen kriterlerin dışında kayıtlara rastlanırsa oluşmaktadır. Bu nedenle içerdiği veriler tablolarda olmayan kayıtlardır. Discard edilen bu kayıtların maksimum sayısı belirtilebilir. Veritabanına yazılan hiçbir kayıt discard dosyalarında yer almaz.

Veri dosyasındaki formatın aynısı ile discard dosyaya atılan kayıtlar yazılır. Bu durumda konrol dosyasında istenilen değişiklikler ya da eklemeler yapıldıktan sonra discard dosyasıda veri dosyası gibi kullanarak, yükleme yapılabilir.

### 6.1.5 Kütük dosyası

SQL*Loader icra edilmeye başladığı anda bir kütük dosyası (log dosyası) yaratır. Eğer log dosyası yaratılmamışsa, sql*loader başlamadan sonlanmıştır. Log dosyası yüklemenin bir özetini teşkil eder.

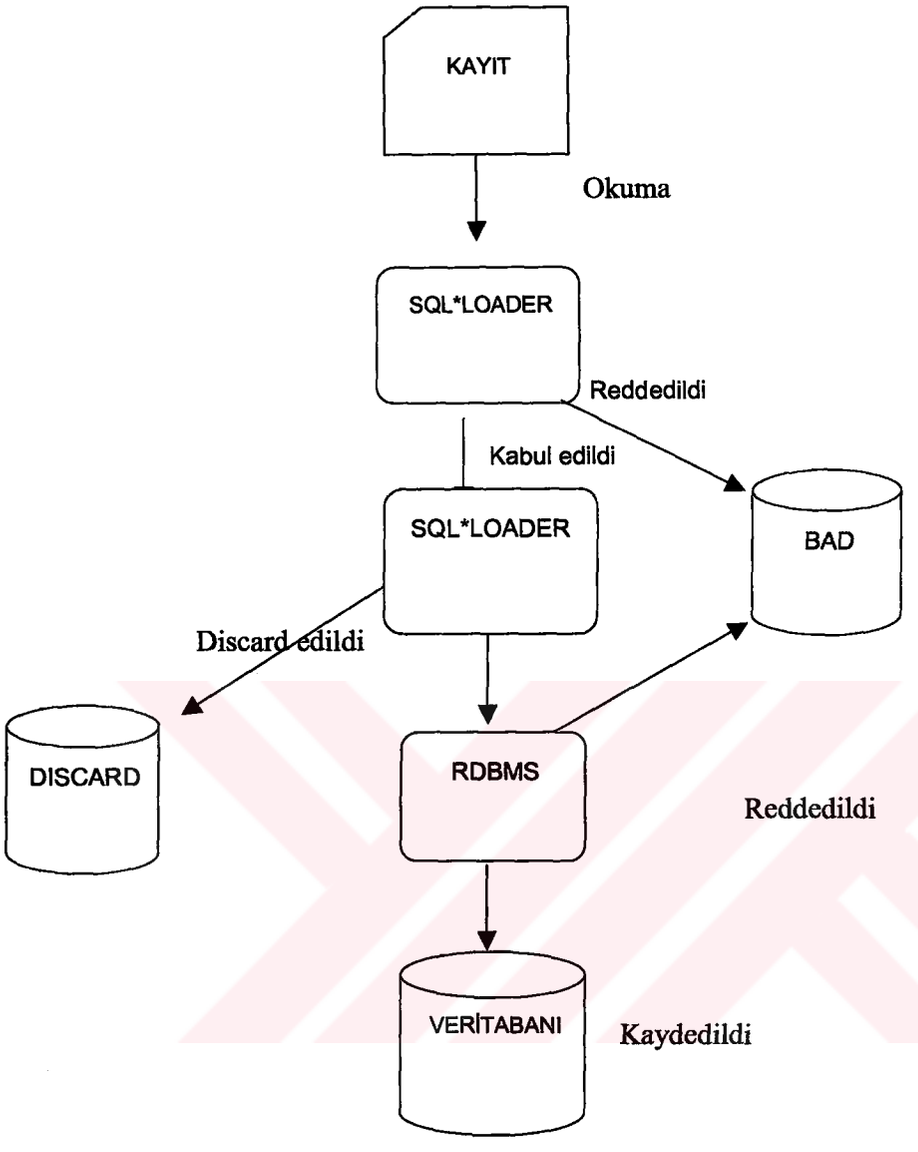
#### 6.1.5.1 Veri yükleme yolları

SQL*Loader iki metodla veri yükler; Geleneksel yol, SQL insert ifadeleri ile bir dizi kullanarak ya da direk yolla verileri veritabanına yükler. Yükleme yapılacak tablolar önceden yaratılmış olarak veritabanında bulunmalıdır., SQL*Loader asla tablo yaratmaz. Tablolar veri içerebilirler ya da boş olabilirler.

Yükleme yapılması için aşağıdaki yetkilere ihtiyaç duyulmaktadır;

- Yüklenecek tablo üzerinde INSERT hakkı.
- Eğer yüklenecek tabloda veri varsa bunların boşaltılması seçilmişse o tablo üzerinde

DELETE hakkı.




Şekil 6.4 Kayıtların filitrelenmesi

**Geleneksel Yol**

Bu tip yüklemelerde giriş kayıtları alan tanımlamalarına göre bölümlere ayrılır (parse edilir) ve her bir veri yerini tutan dizilre kopyalanır. Diziler dolduğunda ya da okunacak başka kayıt kalmadığında dizi kaydı icra edilir.

Sql*Loader LOB alanları dizilere insert yapıldıktan sonra saklar. Bu yüzden eğer işlem sırasında bir hata ortaya çıkarsa LOB alanı boş olur.

Bu yolla yapılan yüklemelerde tablonun özel ihtiyaçları yoktur.

**Direk yol**

Bu yolla yüklemede ilk olarak giriş alanları veri spesifikasyonlarına göre parse edilir, giriş verileri kolon veri tiplerine çevrilir ve kolon dizileri oluşturulmuş olur. Kolon dizileri blok formatlayıcılarına geçirilir. Bunlar Oracle veritabanı bloğu formatında veri blokları yaratırlar.Yeni formatlanan veritabanı blokları bazı ilişkisel veritabanı yönetim sistemi proseslerinden geçerek direk olarak veritabanına yazılır. Direk metot diğerinden daha hızlıdır ama birçok kısıtlamaları vardır.

Direk yol; LOBs, VARRAYs, nesneler ve iç-içe yerleştirilimş (nested) tablolar için kullanılamaz.

**Paraele Direk Yol**

Bu metod eşzamanlı olarak aynı veri segmentine yüklemeyi çoklu direk yollu yükleme oturumuyla izin verir. Direk yoldan daha fazla kısıtlamaları vardır.

Desteklenen koleksiyon tipleri

SQL*Loader iki tip toplamayı destekler;

İç içe tablolar

İç içe tablo; başka tablonun bir kolonu olarak görünen tablo tipidir. Diğer tablolara uygulanabilen tüm operasyonlar bu tabloya da uygulanabilir.

VARRAYs

VARRAYs değişken büyüklükteki dizilerdir. Bir dizi built-in tiplerin sıralanmış grubudur ve element olarak isimlendirilirler. Herbir dizi elementi aynı tiptedir ve bir indeksi vardır ki bu sayı elementin VARRAYs içindeki pozisyonu belirtir. Bir VARRAY yaratilirken, maksimum büyüklüğünün belirtilmesi zorunludur. Bir varray tipi açıklandığında bir ilişkisel tablonun kolonun veri tipi ya da PL/SQL değişkeni olarak kullanılabilir.

Desteklenen LOB Tipleri

LOB; büyük nesne tipidir. Oracle8 dört tip LOB tipini destekler;

- BLOB: yapısal olmayan binary verileri içerir.
- CLOB: tek-byte olan karakter verilerini içerir.
- NCLOB: national character set'inden sabir uzunluktaki karakterleri içerir.

- BFILE: BLOB veritabanının dışında server tarafındaki OS (Operating System – İşletim Sistemi) dosyasında saklanır.

LOB bir kolonun veri tipi olabilir-NCLOB hariç ve ayrıca bir nesnenin attribute veritipi de olabilirler. LOB gerçek değer içerirler, NULL ya da boş olabilirler.

### 6.1.6 Sql*loader komut satırı

SQL*Loader'ı aktif kılacak komut işletim sistemine bağlıdır. Kullanılan örnekler UNIX tabanına göre verilmiştir. "sqlldr"

Geçerli Anahtar Sözcükler:

userid — Oracle username/password

control — Konrol dosyasının Adı

log — Log dosyasının Adı

bad — Bad dosyasının Adı

data – Veri dosyasının Adı

discard — Discard dosyasının adı

discardmax — discard edilecek maksimum kayıt sayısı(varsayılan tümüdür)

skip — Atlanacak mantıksal kayıt sayısı (Varsayılan 0)

load — Yüklenecek mantıksal kayıt sayısı (Varsayılan tümü)

errors — İzin verilecek hata sayısı

rows — conventional path ise bind arraydaki satır sayısı (Varsayılan: Conventional Path 64, Direct path all)

bindsize — Conventional path de bind array ın byte cinsinden değeri (System-bağımlı varsayılan)

silent — Suppress messages during run

(header, feedback, errors, discards, partitions, all)

direct — direct path kullan

(Varsayılan FALSE)

parfile — Parameter dosyası : parameter spesifikasyonlarını belirten dosya

parallel — Paralel load aktif olur (Varsayılan FALSE)

readsize – Okuma bufferının byte cinsinden büyüklüğü

file

### 6.1.6.1 Komut satırı anahtar sözcüklerin kullanımı

Anahtar sözcükler virgülle ayrılırlar. İstenilen sırada girilebilirler. Her birini geçerli argümanlar takip etmelidir.

Örneğin:

SQLLDR CONTROL=foo.ctl, LOG=bar.log, BAD=baz.bad, DATA=etc.dat
USERID=scott/tiger, ERRORS=999, LOAD=2000, DISCARD=toss.dis,
DISCARDMAX=5

### 6.1.6.2 Anahtar Sözcüklerin Kontrol Dosyasında Belirtimi

Eğer komut satırının uzunluğu maksimum değeri aşıyor ise bir kısım komut satırı anahtar Sözcükler kontrol dosyasına OPTION anahtarının altında yazılabilir. Şu argümanları içerebilir;

SKIP = n
LOAD = n
ERRORS = n
ROWS = n
BINDSIZE = n
SILENT = {FEEDBACK | ERRORS | DISCARDS | ALL}
DIRECT = {TRUE | FALSE}
PARALLEL = {TRUE | FALSE}

Örneğin:

OPTIONS (BINDSIZE=100000, SILENT=(ERRORS, FEEDBACK) )

Burada dikkat edilmesi gereken en önemli husus komut satırındaki belirteçler, kontrol dosyasında OPTION altındaki betimlerin geçerliliğini kaldırma üstünlüğüne sahiptir.

Ayrıca argümanlar PARFILE ile belirtilen diğer bir dosyanın içine de yazılabilir.

### 6.1.7 Veri aktarım tipleri ve uygulamaları

Şu tip veri aktarımları SQL*Loader'la yapılabilmektedir;

1. Sabit-Formatlı Alanların Yüklenmesi

2. Değişken-Uzunluktaki Verinin Yüklenmesi
3. Limitsiz, Serbest-Formatlı Dosyaları Yüklenmesi
4. Bileşik Fiziksel Kayıtların Yüklenmesi
5. Çoklu Tablolara Veri Yüklenmesi
6. Direk Yol Yükleme Metodu ile Yükleme
7. Formatlı bir Rapordan Verinin Çıkarılması
8. Bölümlenmiş Tabloya Yükleme
9. LOBFILE ların Yüklenmesi
10. REF alanlarının ve VARRAY ların Yüklenmesi

**Aktarımda Kullanılan Dosya tipleri:**

- Kontrol Dosyaları ( örneğin, yukornek1.CTL )
- Veri Dosyaları ( örneğin, yukornek1.DAT )
- Setup Dosyaları ( örneğin, yukornek1.SQL )

Eğer yüklenecek veri kontrol dosyasında bulunmakta ise bu durumda ayrı bir .DAT dosyası yoktur. Ayrıca eğer özel kurulum adımları yoksa .SQL dosyası da kullanılmaz.

Aşağıdaki Çizelge 6,2 her bir örnek için gereken dosyalar belirtilmiştir.

Çizelge 6.2 Aktarma uygulamalarında kullanılacak dosyalar

| Uygulama | .CTL | .DAT | .SQL |
|---|---|---|---|
| 1 | X | | X |
| 2 | X | X | |

### 6.1.7.1 Sabit-formatlı alanların yüklenmesi

Bu tip yüklemelerde alanların pozisyonları ve tipleri kesin olarak bellidir. Ayrıca kontrol dosyasının dışında bir veri dosyası bulunabilir.

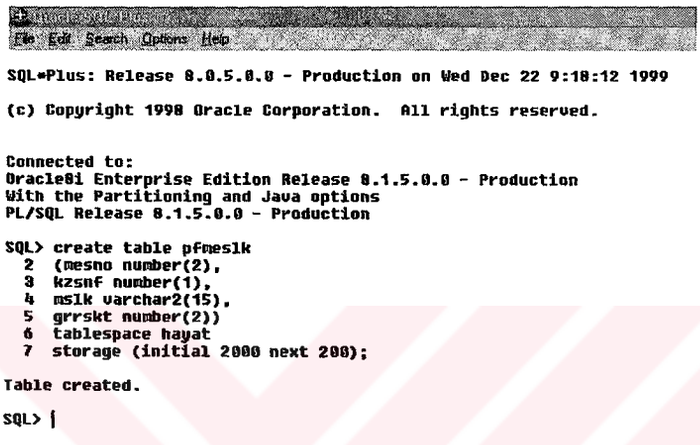
Kontrol dosyamız "YUKORNEK1.CTL" olsun. İçeriği;

```
LOAD DATA
INFILE 'veri_dosya_ismi
INTO TABLE yuklenecek_tablo_ismi
(kolon_ismi_1  POSITION(n:m) KOLON_TIPI,
kolon_ismi_2  POSITION(n:m)  KOLON_TIPI,
..... )
```

LOAD DATA : Kontrol dosyalarının başlangıcında ihtiyaç duyulan bir komuttur.
INFILE: Aktarılacak veriyi içeren dosya isminden önce yazılır.
INTO TABLE dan sonra yazılan tabloya veri aktarımı gerçekleşir.
4. ve 5. satırlarda verinin aktarılacağı kolon ismi ve verinin veri dosyasındaki konumu belirtilir.

### 6.1.7.2 Uygulama

Veri yüklenecek tablo önceden yaratılmalıdır. CREATE TABLE komutu ile PFMSLK adlı bir tablo yaratılsın.



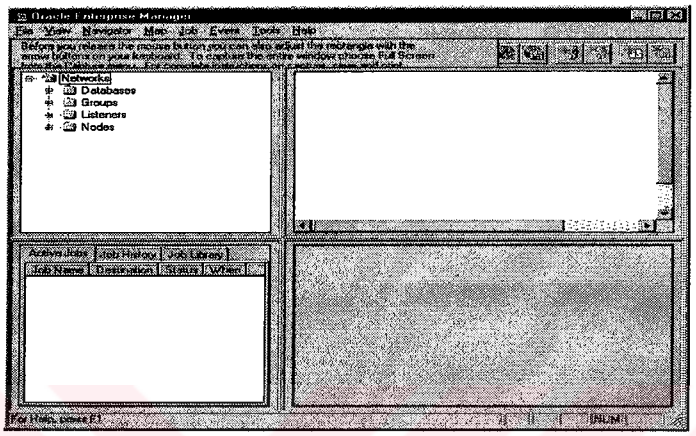

Şekil 6.5 Tablo yaratımı (Oracle SQL*Plus)

Yaratılan bu tabloya veri aktarımını sağlayan kontrol dosyası sabit-formatlı alanların yükleneceği baz alınarak düzenlenmiştir. Kontrol dosyası içeriği;

```
LOAD DATA
INFILE 't:\data\pfmeslk.txt'
BADFILE 't:\bad\hayat.bad'
DISCARDFILE't:\dis\hayat.dis'
REPLACE
INTO TABLE PFMESLK
(MESNO   POSITION(03:04)   INTEGER EXTERNAL,
 KZSNF   POSITION(06:06)   INTEGER EXTERNAL,
 MSLK    POSITION(07:21)   CHAR,
GRRSKT   POSITION(24:25)   INTEGER EXTERNAL)
```

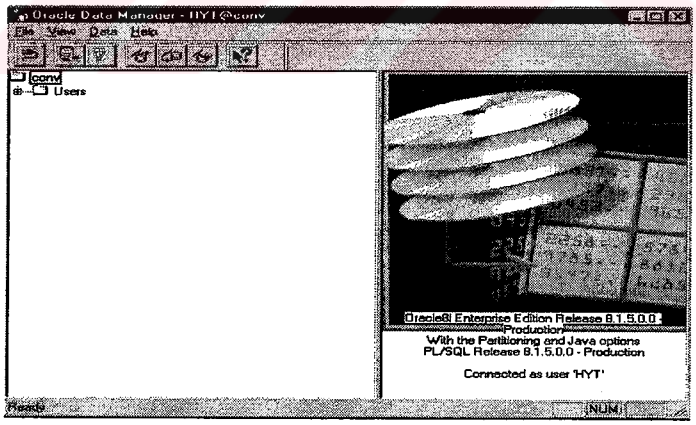
Yükleme işlemini Oracle Enterprise Manager'dan Oracle Data Manager Tool ile yapılabilmektedir. İşlemler şu şekilde ilerler;

1) Şekil 6.6'daki Oracle Enterprise Manager'da Tools menüsünden Oracle Data Manager seçilir.



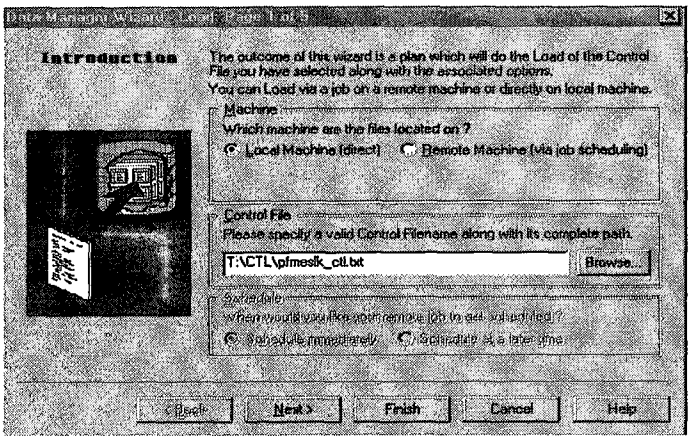

Şekil 6.6 Yükleme işlemi ilk adımı (Oracle Enterprise Manager)

2) Data menüsünden Load seçilir ve yükleme işlemi spesifikasyonları ilerki aşamalarda belirtilir.(Şekil 6.7)



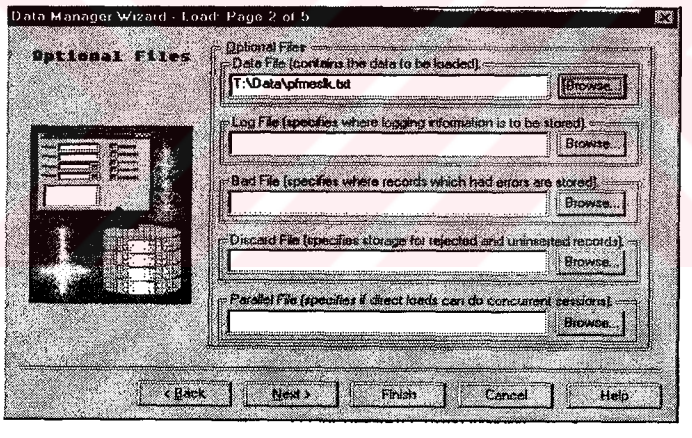

Şekil 6.7 Yükleme işlemi ikinci adımı (Oracle Data Manager)

3) Yüklemenin birinci adımında kontrol dosyalarının bulunduğu yer belirtilir.Dosya çalıştığımız makinede olabileceği gibi ağdaki başka bir makine de olabilir, bu doğrultuda Local Machine ya da Remote Machine işaretlenir (Şekil 6.8).




Şekil 6.8 Kontrol dosyası belirtimi (Oracle Data Manager)

4) Sonraki adımla veri dosyasının (yüklenecek veriyi içeren dosya), log dosyasının ( log bilgilerini içeren dosya), bad dosyası (hatalı kayıtların yerini belirtir), discard file (rededilen veya kayıt edilemeyen kayıtları saklayan dosya) ve parallel dosyasının (eş zamanlı olarak yüklenecek dosya) seçimlik olarak belirtilir. Eğer bu dosyalar kontrol dosyasında belirtilmiş ise tekrarlanmasına gerek yoktur. (Şekil 6.9)

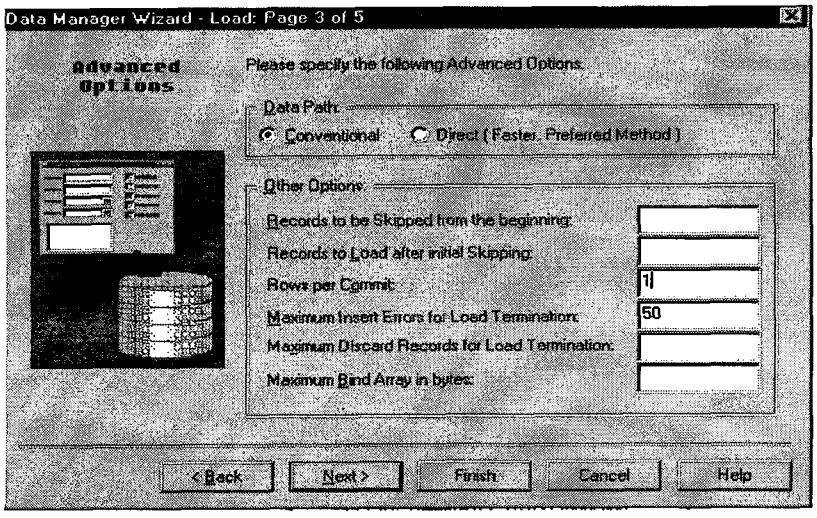



Şekil 6.9 Seçimli dosya seçimi (Oracle Data Manager)

5) Şekil 6.10'de veriyi yükleme tipi belirtilir. Geleneksel ya da direk olarak iki seçenek mevcuttur. Direct daha hızlı olup tercih edilen yoldur. Ayrıca aşağıdaki kısılar koyulabilir;

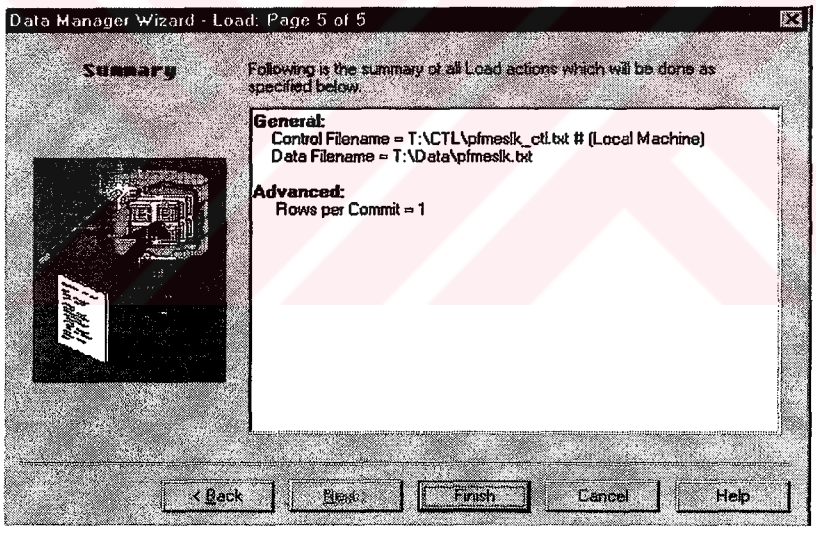
- Başlangıçtan itibaren atlanacak kayıt sayısı
- İlk atlamadan sonra yüklenecek kayıt
- Yüklenecek verinin kaç satırda bir onaylanacağı
- Kayıtta yapılabilecek en çok hata sayısı
- Discard edilebilecek en büyük kayıt sayısı

- Eğer geleneksel yol seçildiyse byte cinsinden maksimum dizi genişliği verilir.



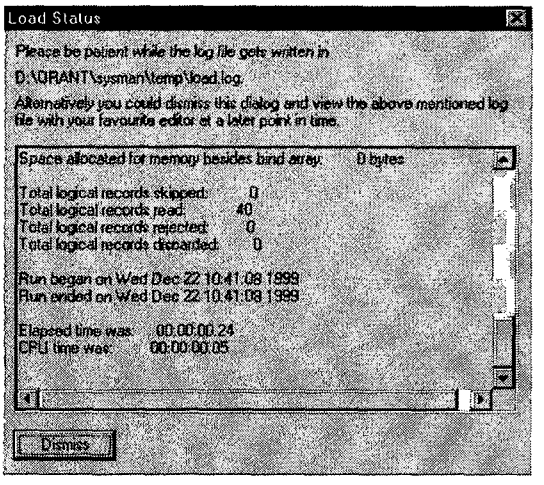

Şekil 6.10 Veri yolu seçimi (Oracle Data Manager)

6) Şekil 6.11 yüklemenin son penceresidir. İşlemin özetini gösterir.




Şekil 6.11 Veri yükleme özeti (Oracle Data Manager)

7) Şekil 6.12'de yüklemenin nasıl sonuçlandığı gösterilir.Bu pencere dismiss edilerek log dosyasından yüklemenin durumu kontrol edilebilir. Kapatılmazsa log dosyası ekranda görüntülenir.




Şekil 6.12 Veri yükleme sonuçları (Oracle Data Manager)

Sabit formatlı verinin Oracle veritabanındaki tabloya aktarılması işleminin sistemde oluşan log dosyası Ek 2'de yer almaktadır.

Log dosyası yükleme işlemindeki tüm adımları özetler. Dosyanın başında, yüklemede kulanılacak Kontrol dosyası, veri dosyası, bad dosyası ve discard dosyası isimleriyle beraber lokasyonları da belirtilerek yer alır.

Yüklenen veri sayısı herhangi bir kısıtlama koyulmadığı için "ALL" dur. Atlanan veri olamadığı için sıfırdır, izin verilen hata sayısı değişiklik yapılmadığından varsayılan 50 alınmıştır. Dizi genişliği de varsayılan 65536 byte dır. İlişkisel yol kullanılmıştır.

PFMESLK tablosuna her mantıksal kayıttan yükleme yapılmış ve kayıt giriş seçimi de kontrol dosyasında belirlendiği üzere REPLACE alınmıştır. Yani yükleme öncesinde tabloda veri varsa silinir.

Kontrol dosyasında belirtilmiş sabit formatlı kolonlar okunarak yükleme başlar. Sonucunda 40 satır başarıyla yüklenmiştir. Veri hatasından yüklenmeyen kayıt yoktur. When klozundan dolayı yüklanmeyen kayıtta sıfırdır ve alanların null olmasından dolayı yüklenemeyen satır da bulunmamaktadır.

28 byte dizi genişliği için tutulmuştur ama hafıza için yer ayrılmamıştır.

Atlanan toplam mantıksal kayıt = 0

Okunan toplam mantıksal kayıt = 0

Rededilen toplam mantıksal kayıt = 0

Atlanan toplam mantıksal kayıt = 0

Log dosyasının sonun yüklemenin başlangıç ve bitiş tarihleri gün,ay,saat ve yıl bazın , yüklemenin toplam zamanı ve cpu zamanı gösterilir.

### 6.1.7.3 Değişken-uzunluktaki verinin yüklenmesi

Kontrol dosyası yüklenecek olan tablonun ve kolonlarının tanımlanmasını içerebilir. Bu durumda veri dosyasına gerek kalmaz. Kontrol Dosyası içeriği;

```
1. LOAD DATA
2. INFILE *
3. INTO TABLE table_name
4. FIELD TERMINATED BY ',' OPTIONALLY ENCLOSED ' " '
5. (column_name1,column_name2,...)
6. BEGINDATA
7. .....
```

1. LOAD DATA ; kontrol dosyalarının başlangıcında ihtiyaç duyulan bir belirteçtir.
2. INFILE * ile verinin ; dosya dışındaki bir dosyadan alınması yerine kontrol dosyasının içinden bulunacağını gösterir.
3. INTO TABLE ile yüklemenin yapılacağı tablo belirtilir. SQL*Loader boş bir tabloya ihtiyaç duyar.
4. FIELD TERMINATED BY; ile verileri birbirinden ayıran işaret belirtilir. Bu işaret kayıtların içinde bulunmalıdır. Seçimlik olarak, OPTIONALLY ENCLOSED ile başka bir ayraç daha tanımlanabilir.
5. Parantez içinde yüklenecek kolon isimleri belirtilir. Eğer kolon tipi belirtilmezse varsayılan olarak 255 uzunluğunda char alınır.
6. BEGINDATA verinin başlangıcını belirtir.

### 6.1.7.4 Uygulama

```
1 create table subacnpf
2 (bnkkd number(3) not null,
3 sbkod number(4) not null,
4 acent number(5))
5 tablespace hayat
6* storage (initial 2000 next 200)
SQL> /
```

Table created.

Tablo yaratıldıktan sonra kontrol dosyası oluşturulur. Değişken uzunluktaki yüklenecek veriler kontrol dosyasının sonunda BEGINDATA komutundan sonra yer alır.

Alanların sonralarında ayraç olarak ',' olacağı belirtilmiştir. Istenilirse " ile de alanlar ayrılabilir.

SUBACNPF1_CTL kotrol dosyası içeriği;

```
LOAD DATA
INFILE *
INTO TABLE SUBACNPF
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
(BNKKD,SBKOD,ACENT)
BEGINDATA
2,1459,10219
2,1465,10045
2,1477,11251
2,1480,11013
2,1486,10045
2,1491,10369
2,1497,10239
2,1501,11464
2,1506,10144
2,1507,11152
.....
```

**DOS Ortamında Yükleme İşlemi:**

sqlldr80 conrol=subacnpf1_ctl log=subacnpf1_log bad=subacnpf1.bad direct=true userid=hyt/hyt@conv errors=50 skip=0

subacnpf1_sql dosyasının aktif edilmesi ile yükleme başlar.

SQL*Loader'ın çalışmasının sonlanması ile oluşan log dosyası ise Ek 3'de yer almaktadır.

SQL*Plus 'dan yüklediğimiz tablonun tanımını ve yüklenen verileri kontrolü şu şekildedir yapılır;

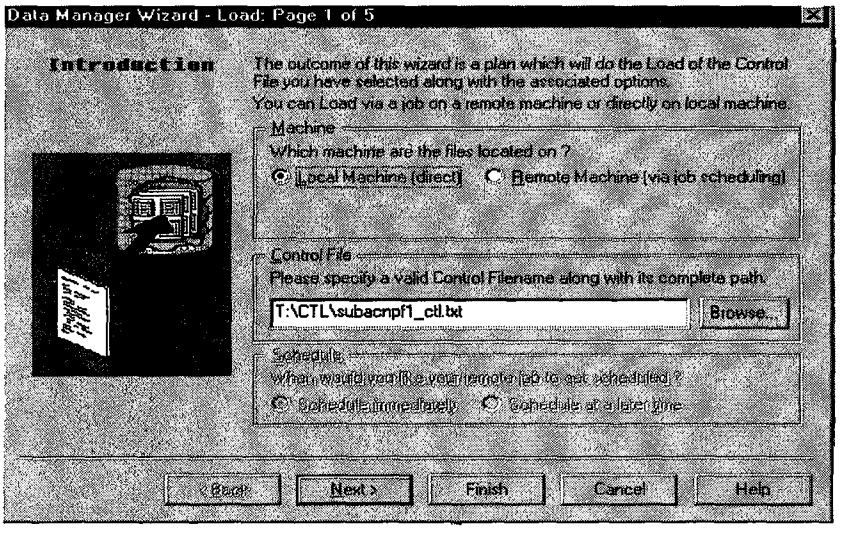
```
SQL> DESC SUBACNPF1;
```

```
Name                    Null?   Type
----------------------- ------- ----
BNKKD                   NOT NULL NUMBER(3)
SBKOD                   NOT NULL NUMBER(4)
ACENT                            NUMBER(5)

SQL> SELECT * FROM SUBACNPF1;

   BNKKD    SBKOD    ACENT
-------- -------- --------
       2     1459    10219
       2     1465    10045
       2     1477    11251
       2     1480    11013
       2     1486    10045
       2     1491    10369
       2     1497    10239
       2     1501    11464
       2     1506    10144
       2     1507    11152
.....
.....

75 rows selected.
```

### 6.1.7.5 Belirli bir aralıktaki verilerin aktarımı

Belirli aralıklı verilerin Oracle veritabanına aktarılması mümkündür. Aşağıdaki uygulama metin dosyasındaki kayıtların birincisini atlayarak on kaydın veritabanına aktarımına örneği teşkil eder.

1) Şekil 6.13'de Subacnpf1_ctl kontrol dosyasının yeri belirtilir.

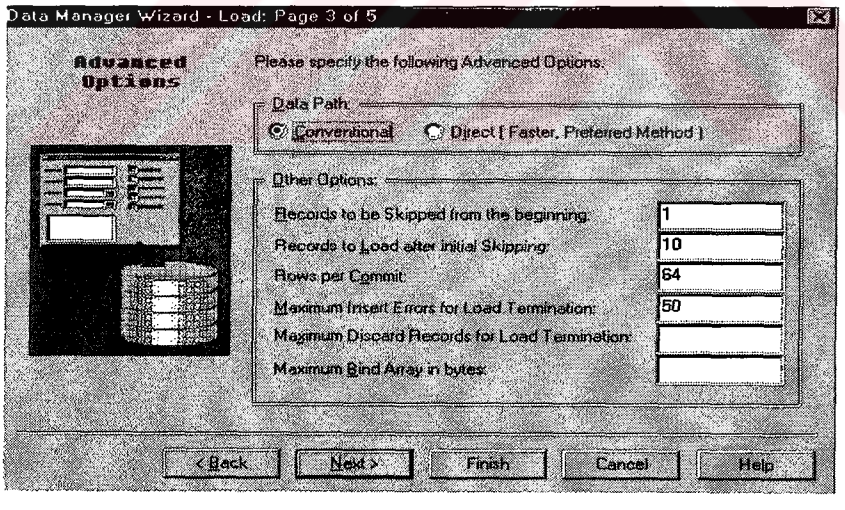



Şekil 6.13 Kontrol dosyası seçimi (Oracle Data Manager)

2) Şekil 6.14'de atlanacak kayıtların lokasyonları belirtilir.

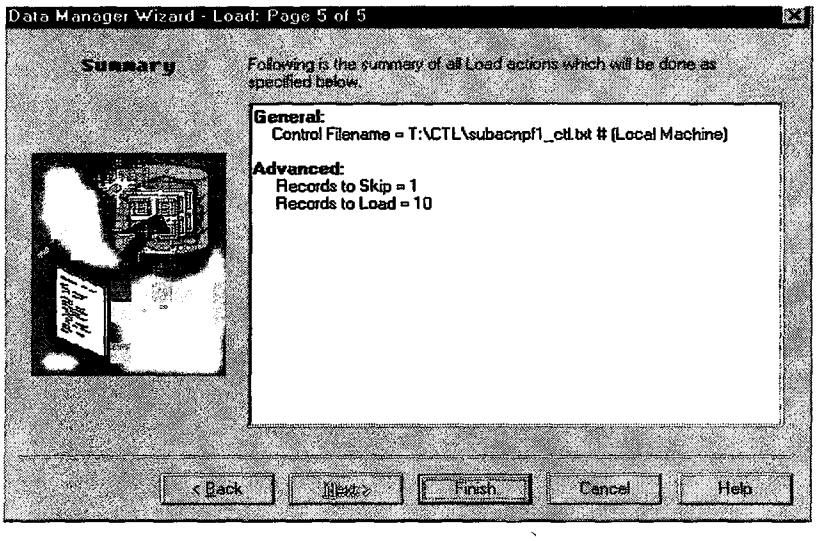
**"Records to be Skipped from the beginning:"** seçeneği ile başlangıçtan itibaren atlanacak olan kayıt sayısı belirtilir. Bu örnekte 1 tane kayıt atlanacaktır.

**"Records to Loads after initial Skipping:"** seçeneği ile de bir önce ki seçeneğin belirttiği kayıttan itibaren yüklenecek olan kayıt sayısı burada verilir.



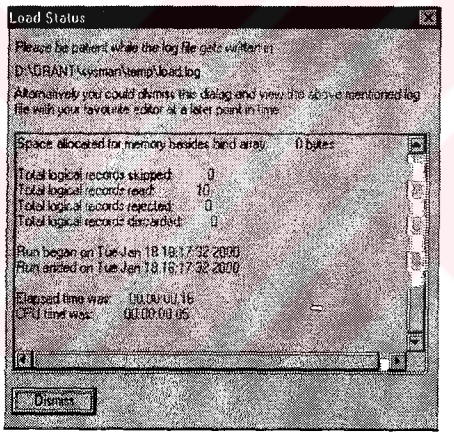

Şekil 6.14 Aktarımın seçeneklerinin belirtimi (Oracle Data Manager)

3) Şekil 6.15'de yükleme parametrelerin özeti sunulur.




Şekil 6.15 Veri yükleme özeti (Oracle Data Manager)

4) Eğer bu spesifikasyonlar onaylanırsa yükleme başlatılır ve tamamlandığında aşağıdaki "Load Status" penceresi son durumu verir.




Şekil 6.16 Veri yükleme sonuçları (Oracle Data Manager)

Yükleme sonucu oluşan log dosyası Ek 4'de yer almaktadır.

Verilerin okunduğu veri dosyası: subacnpf1.data şu formatadır.

2 1459 10219

2 1465 10045

2 1477 11251

2 1480 11013

2 1486 10045

```
2 1491 10369

2 1497 10239

2 1501 11464

2 1506 10144

2 1507 11152

2 1508 10966

2 1513 10817

2 1518 10771

.....
```

Bu veri dosyasından (data file) bir kayıt atlanıp on tane kayıt seçilerek yükleme yapılmıştır. Sonuçta aktarılan verinin bulunduğu subacnpf1 tablosunun son haline SQL*Plus yardımıyla bakalım.

```
SQL> select * from subacnpf1;
```

| BNKKD | SBKOD | ACENT |
|---|---|---|
| 2 | 1465 | 10045 |
| 2 | 1477 | 11251 |
| 2 | 1480 | 11013 |
| 2 | 1486 | 10045 |
| 2 | 1491 | 10369 |
| 2 | 1497 | 10239 |
| 2 | 1501 | 11464 |
| 2 | 1506 | 10144 |
| 2 | 1507 | 11152 |
| 2 | 1508 | 10966 |

10 rows selected.

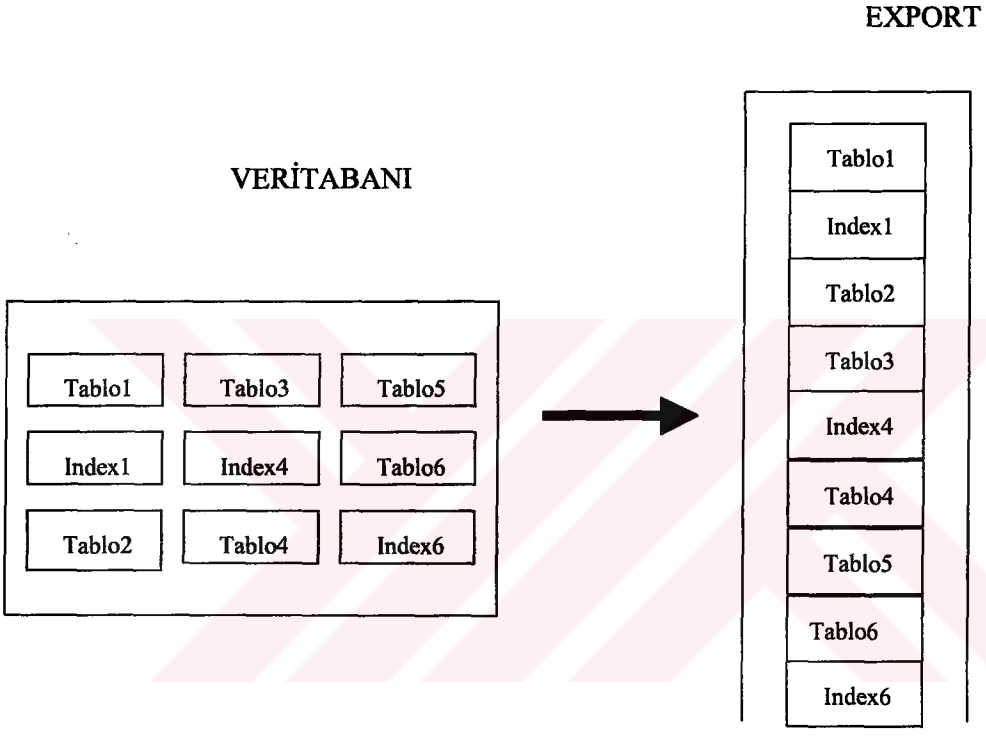
## 6.2 Export

Export, veri nesnelerini Oracle veritabanları arasında yazılım ve donanım açısından farklı platformda olsalar dahi transfer etmeyi sağlayan bir Oracle hizmetidir. Export nesne tanımlarını ve tablo verilerini Oracle veritabanından çıkarır ve onları ikili formattaki bir Export dump dosyasına atarak bir disk veya bir teypte saklar. Bu tip dosyalar FTP edilebilir.

Ayrıca Export yedeklemenin bir diğer alternatifi olarak da kullanılabilir.

Exportla transfer edilen dump dosyaları yalnızca diğer Oracle hizmeti olan Import tarafından okunabilir. ASCII sabit-formatta ya da limitsiz dosyalarda yüklü veriyi okumada da SQL*Loader kullanılmaktadır (Oracle, 1999g).

Export, Şekil 6.17'de gösterildiği gibi Oracle veritabanındaki nesneleri (örneğin tablolar) kendilerine bağlı nesnelerle ( örneğin indeksler, grantler) veritabanından çıkarılabilir.




Şekil 6.17 Nesnelerin veritabanından çıkarılması

### 6.2.1 Ulaşım İmtiyazı

Export'u kullanmak için Oracle veritabanında CREATE SESSION imtiyazına sahip olmak şarttır. Diğer kullanıcıların sahip olduğu tabloların export'unu almak için EXP_FULL_DATABASE rolüne sahip olmak da gerekir. Bu rol tüm DBA'lere atanmıştır.

Eğer EXP_FULL_DATABASE sistem imtiyazına sahip olmayan bir kullanıcı başka bir kullanıcının şemasından hiçbir nesneyi export edemez. Eşanlam(Synonym) yaratılmış olsa

dahi export alınamaz. Aşağıdaki şemalar veritabanı tarafından korunmuştur, export işlemi uygulanamazlar.

- ORDSYS
- MDSYS
- CTXSYS
- ORDPLUGINS

CATEX.SQL ya da CATALOG.SQL veritabanı yaratıldıktan sonra çalıştırılmalıdır. Bu scriptler gerekli export görüntülerini yaratırlar, gerekli imtiyazları EXP_FULL_DATABASE rolüne atar, EXP_FULL_DATABASE'i DBA rolüne atarlar.

Export işlemi başlatılmadan önce depolanacak alanda -diskte ya da teypte yeterli yer kontrolü yapılmalıdır. Yeterli yer yoksa Export yazım-hatası ile sonlanır. Tabloların büyüklükleri ihtiyaç duyulacak maksimum yer için bilgi verir. Bu veriler data dictionary'nin bir görüntüsü olan USER_SEGMENTS tablosundan bulunabilir.

select sum(bytes) from user_segments where segments_types = 'TABLE';

### 6.2.2 Export tipleri

Export işlemi şu şekillerde yapılabilir.

1) Komut satırından,

exp username/password PARFILE=filename

PARFILE export parametrelerinin içeren dosyadır. Farklı veritabanları için farklı parametreler kullanılıyorsa, farklı parametre dosyaları oluşturulabilir.

2) Aşağıdaki komut satırıyla;

exp username/password

Bu satırda ihtiyaç duyulan parametreler yazılabilir.

3) exp username/password komutu girişi ile bire birebir oturum başlar. Bu metot diğerlerine

oranla daha az fonksiyoneldir.

Birinci ve ikinci metot beraber olarak kullanılabilir. Yani komut satırında parametre dosyası ile parametreler yer alabilir. Ayrıca aynı parametre hem parametre dosyasında olabilir hem de parametre olarak yer alabilir. Bulunduğu pozisyon ise hangisinin diğerinin hükmünü ortadan kalktığını belirler. Örneğin; param.dat dosyası INDEXES=Y parametresini içersin ve komut satırı şu şekilde düzenlensin.

exp system/manager PARFILE=param.dat INDEXES=N

INDEXES=N , PARFILE=param.dat dan sonra yazılmıştır ve parametre dosyasının içindeki

YES seçiminin hükmünü bitirir.

Ayrıca kullanıcı adı ve şifre de parametre dosyasına yazılabilir ama güvenlik nedeniyle önerilmeyen bir metottur.

#### 6.2.2.1 Komut satirindaki parametreler

EXP <parametre> = (<değer>, <değer>,...)

Komut satırında USERID ilk parametre olmak zorundadır. Diğer parametreleri, tanımları ve ilk değerleri şunlardır;

- USERID : Kullanıcı_adı/şifre
- FULL : Tüm veritabanı expotu belirtimi için kullanılır. Değerleri; Y ya da N olabilir.
- BUFFER: Tampon büyüklüğünü belirtir.
- OWNER : Export edilecek verilerin sahip kullanıcıların ismi listesi
- FILE : Dump dosyası adıdır ve uzantısı .dmp olmak zorundadır. (EXPDAT.DMP)
- TABLES : Export edilecek tabloların listesi.
- COMPRESS : Yalnızca bir genişlemeye export alınmasını belirtir. Değerleri Y ya da N dir.
- RECORDLENGTH : I/O kaydının uzunluğunu belirtir.
- GRANTS : Kullanıcıların haklarının da export edilmesini belirtir. İlk değeri Y dir.
- INCTYPE : Export işleminin artış tipini belir. COMLETE, CUMULATIVE ya da INCREMENTAL modlarda export alınabilir.
- INDEXES : İndekslerinde export edilmesini belirtir. İlk değeri Y dir.
- RECORD : Incremetal export işleminin kayıt bazlı yapılmasını sağlar.
- ROWS : Tablo kayıtlarının export edilmesini belirtir. İlk değeri Y dir.

- PARFILE : Tüm parametreleri içerebilen metin dosyalarıdır.
- CONSTRAINTS : Kısıtlamaların export'unun alımasını sağlar. İlk değeri Y)
- CONSISTENT : Tablolar arası ilişkinin bozulmamasını sağlar.
- LOG : İşlemlerinin logunun tutulacağı dosyadır.
- STATISTICS : Export esnasındaki gerekli optimizasyonların tipleri belirtilebilir. Değerleri COMPUTE, ESTIMATE ya da NONE dır.
- DIRECT : Rollback segmentlere yazılımını sağlar. İlk değeri N dir.
- TRIGGERS : Tetikleyicilerini export edilmesini belirtir.
- FILESIZE : dump dosyalarının maksimum büyüklüğü kısıtlama olarak verilebilir.
- QUERY : Tablodaki veriler sorgulanarak export edilebilir.

### 6.2.3 Export işlem tipleri

Tabloların verilerinin export edilmesi için iki yolla yapılabilir.

1) Geleneksel yolla Export

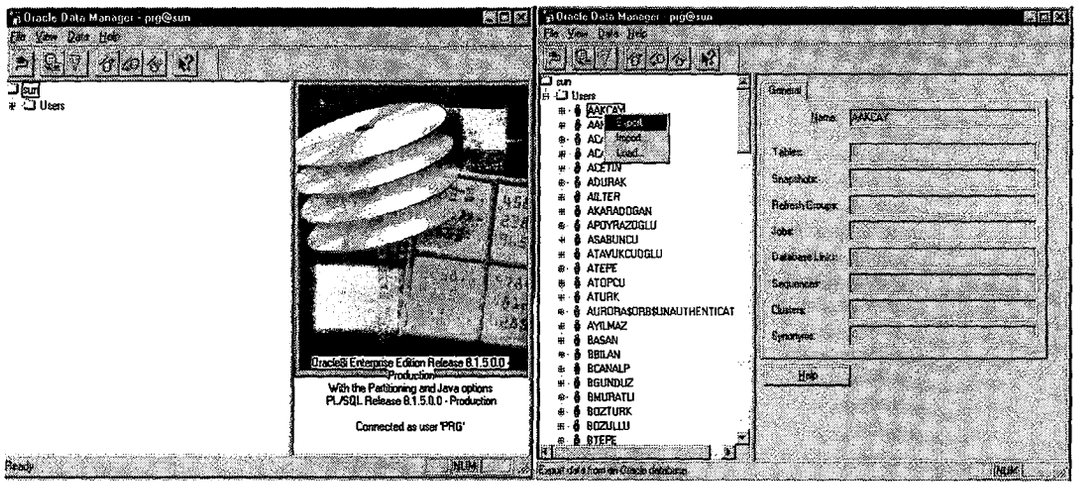
2) Direk yolla Export

Geleneksel yolda , SQL SELECT ifadesini kullanarak verileri tablolardan çıkarılır. Veri diskten tampon cache'e okunur ve satırlar değerlendirilme tamponuna transfer edilir. Buradan da istemci tarafına transfer edilerek export dosyalarına yazılırlar.

Direk yolla ise Export daha hızlıdır. Veri diskten tampon cache'e okunur ve satırlar direk olarak istemciye alınır. Değerlendirme tampon bölgesi atlanır. Export'un beklediği formatta olan veriler için değerlendirme tampon bölgesinde yapılan gereksiz veri dönüşümleri elimine edilmişolur. İstemci tarafından transfer olan veriler direk olara export dosyasına yazılır.

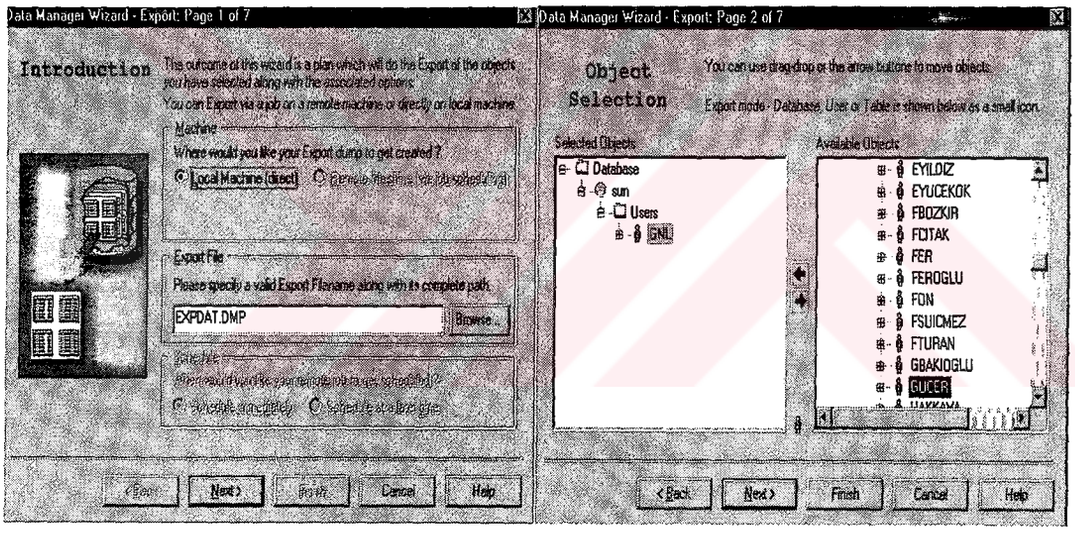
### 6.2.4 Oracle'da veri çıkarımı

Oracle Data manager ile tüm veritabanının export'u alınabildiği gibi kullanıcı bazında da export alınabilir. Bu uygulamada sun veritabanına bağlı olan GNL kullanıcısının export'u alınacaktır (Şekil 6.18).



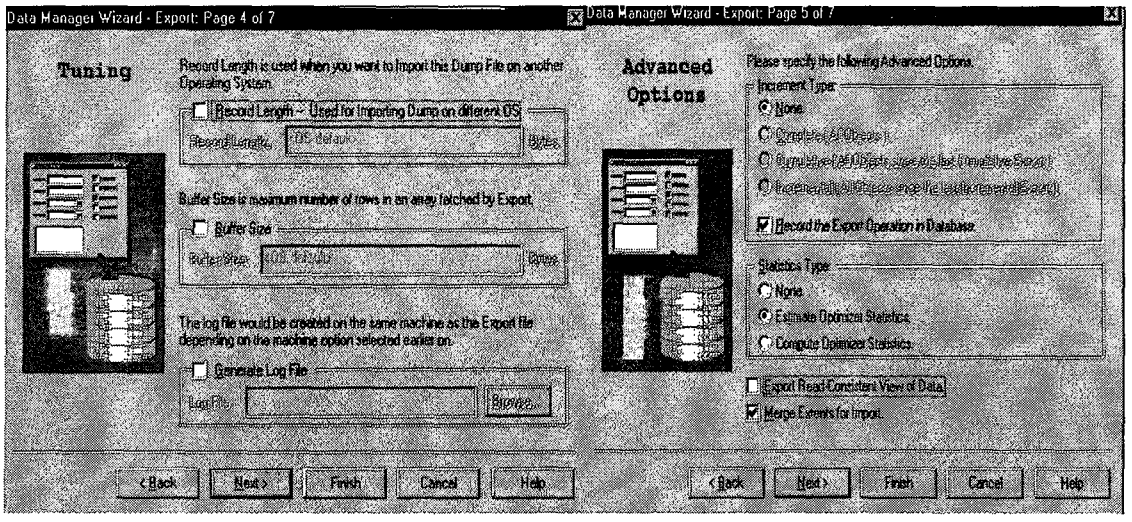

Şekil 6.18 Export için veritabanı ve kullanıcı seçimi (Oracle Data Manager))

Dump dosyası lokaldeki makinede saklanabildiği gibi başka bir makineye de atılabilir. Ayrıca dump dosyasının uzantısı DMP olmak zorundadır (Şekil 6.19).

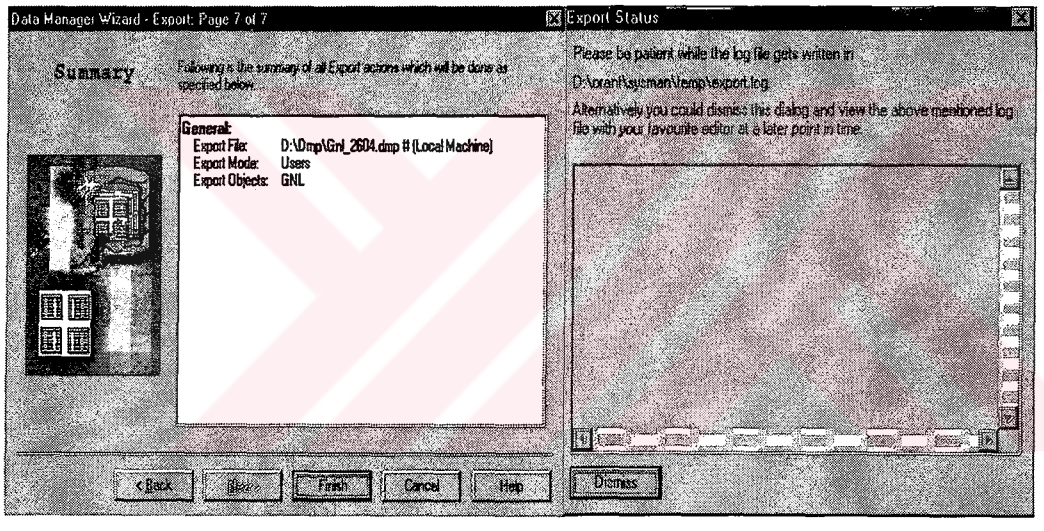



Şekil 6.19 Export dump dosyası seçimi (Oracle Data Manager)

Şekil 6.20'de kayıt tampon alan büyüklüğü belirtilebilir. Logların tutulması için log dosyası seçilir.




Şekil 6.20 Export ayar ve seçenekleri (Oracle Data Manager)




Şekil 6.21 Export sonuçları (Oracle Data Manager)

Export sonucu oluşan export log dosyası Ek 5'de yer almaktadır:

Aynı export işlemini komut satırında çalışması için oluşturulan parametre dosyası şu formattadır;

USERID=prg/prg@sun

FILE=D:/Dmp/Gnl_2604.dmp # (Local Machine)

OWNER=(GNL)

LOG=D:\orant\sysman\temp\export.log

## 6.3 Import

Import, bir veritabanından Export ile çıkarılan veri nesnelerini kaydeder. Export'un çıkardığı verilerin bulunduğu dump dosyası yalnızca Import tarafından okunabilir (Oracle, 1999g).

Import nesne tanımlarını ve Export'la çıkarılan tablo verilerini ikili formattaki dump dosyasından okur.

Tablo verileri Export dosyasından okunarak import edilir. Export dosyası sırasıyla aşağıdaki nesneleri içerir.

1) Tip tanımları
2) Tablo tanımları
3) Tablo verileri
4) Tablo indeksleri
5) Bütünlük kısıtlamaları, görüntü, prosedürler ve tetikleyiciler
6) Bitmap, fonksiyonel ve alan indeksler.

Import işleminde ilk olarak yeni tablolar yaratılır. Ardından veri import edilir ve indeksler kurulur. Bir sonraki aşamada tetikleyiciler import edilir, bu yeni tabloların üzerine bütünlük kısıtları oluşturulur ve son olarak diğer bitmap, fonksiyonel ve alan indeksler kurulur. Bu sıralama verilerin import edilirken geri dönmesini engeller.

Örneğin; EMP isimli bir tablonun üzerinde DEPT tablosuna bütünlük kısıtlaması olsun. Eğer EMP ilk olarak import edilirse, EMP tablosunun referans aldığı DEPT tablosu henüz import edilmediği için ve kısıtlama geçerli olduğundan EMP verileri geri çevrilecektir.

Bu nedenle import işlemi yapılmadan otomatik olarak kısıtlamalar geçersiz kılınır ve aktarmadan sonra tekrar kısıtlamalar aktif edilir, bu şekilde sorun çıkmayacaktır.

## 6.4 Import Tipleri

Import hizmeti dört tip import işlemine olanak tanır. Nesneler import edilirken import modları ve export edilmiş modları gözönüne alınarak import edilir. Bu işlemi yapacak kullanıcının IMP_FULL_DATABASE rolüne sahip olması gerekmektedir.

Import işlemi tipleri;

1) Tablo : Tablo bazında export belirlenen tabloların ya da tablo parçalarının import edilmesine olanak sağlar.

2) Kullanıcı: Kullanıcıya ait tüm nesnelerin (indeks,porsedür gibi) import edilmesine olanak tanır..

3) Veritabanı : Yalnızca IMP_FULL_DATABASE rolüne sahip kullacılar tüm veritabanının export alındığı dump dosyalarını import edebilirler.

4) Transfer edilebilen Tablespaceler : Yetkili kullanıcılar tarafından tablespaceler bir veritabanından diğerine aktarılabilir.

Import işlemi öncesi, veritabanı kurulumundan sonra CATEXP.SQL ya da CATALOG.SQL scriptleri çalıştırılırmalıdır.

Diğer yapılması gereken işlemler;

1. IMP_FULL_DATABASE rolüne gerekli imtiyazlar verilir.

2. IMP_FULL_DATABASE, DBA'a atanır.

3. Gerekli olan görüntü ve veri sözlükleri yaratılır.

### 6.4.1 Import yolları

Üç tip import yapılabilir.

1. imp kullanıcı_adı/şifre PARFILE=dosya_adı

PARFILE dosyası import parametrelerini içerir.

2. imp kullanıcı_adı/şifre <parametreler>

3. imp kullanici_adi/şifre

bu tip imprt işleminde gerekli parametreler bire bir kullanıcıdan alınır.

### 6.4.2 Import parametreleri

- USERID : Kullanıcı_adı/şifre
- FULL : Dump dosyasına tüm veritabanı expotu alınmış ise FULL = Y kullanılır.

FULL parametresi aşağıdaki tüm sistem nesnelerini içerir.

- Profiller

- Public veritabanı bağlantıları
- Public synonym
- Roller
- Rollback Segment Tanımlamaları
- Resource costs
- Foreign Fonksiyon Kütüphaneleri
- Context nesneleri
- Sistem Prosedürel nesneler
- Sistem Audit Seçenekleri
- Sistem İmtiyazları
- Tablespace tanımları
- Tablespace kotaları
- Kullanıcı tanımları
- Dictionary diğer adları
- Sistem Event Triggerları

- BUFFER : Ilk değeri işletim sistemine bağlıdırByte cinsindendir. Buffer büyüklüğü importla insert edilen dizilerdeki satır sayısını belirler. Aşağıdaki formül verilen dizideki satırların insert'i için gerekli yaklaşık buffer büyüklüğünü verir.

buffer_size = rows_in_array * maximum_row_size

LONG,LOB,BFILE,REF,ROWID,DATE ya da tip kolonları içeren tablolar için satırları tek tek insert edilir. Buffer'ın büyüklüğü LOB ve LONG kolonları hariç tüm satın büyüklüğünden daha çok olmalıdır. Eğer buffer satırı tutamazsa Import daha çok buffer işgal der.

- FROMUSER : Import edilecek nesnelerin sahiplerinin listesini belirtir.
- FILE : Export alınırken oluşan dump dosyasının yeri ve adı belirtilir.
- TOUSER : Import işleminin hangi kullanıcıya yapılacağını belirtir.
- TABLES : Import edilecek tabloların listelendiği parametredir.
- IGNORE : Import sırasında varolan tablolarla karşılaşıldığında yaratılmaması IGNORE= Y ile sağlanır.

- RECORDLENGTH : I/O kaydının uzunluğunu belirtir.
- GRANTS : Kullanıcıların haklarının da import edilmesini belirtir. İlk değeri Y dir.
- INCTYPE : Impotum işleminin artış tipini belir. SYSTEM ya da RESTORE değerlerini alabilir.
- INDEXES : İndekslerinde import edilmesini belirtir. İlk değeri Y dir.
- COMMIT : İlk değeri N dir. Her bir satırın kaydı sonrasında onay (commit) işlemi yapılmalı mı sorusunun cevabını verir. Varsayılan olarak Import her bir tablonun yüklenmesinden sonra commit eder ve bir hata ortaya çıktığında bir sonraki nesne ile devam etmeden önce rollback eder.

Eğer COMMIT = N ve import edilen tablo bölümlenmişse (partitioned), herbir bölüm ve alt bölüm ayrı bir transactionda (işlemde) import edilir.

COMMIT=Y rollback segmentlerin büyümesini engeller ve çok büyük nesnelerin import edilme performansını artırır. Eğer tekil kısıtlaması olan bir tablo import edilecekse COMMIT=Y seçimi önerilir. Eğer tekrar import başlatılırsa henüz kayıtlı olan satırlar geri çevrilir ama import sürer. Eğer tabloda unique kısıtlaması yoksa ve import tekrar yapılırsa çift satırlar oluşur. Eğer tabloda LONG,LOB,BFILE,REF,ROWID,UROWID,DATE ya da tip kolonları varsa dizi şeklinde insert yapılmaz. Eğer COMMIT=Y ise bu tablolarda her bir satırdan sonra commit edilir.

- ROWS : Tablo kayıtlarının import edilmesini belirtir. İlk değeri Y dir.
- PARFILE : Tüm parametreleri içerebilen metin dosyalarıdır.
- LOG : İşlemlerinin logunun tutulacağı dosyadır.
- CONSTRAINTS : Tabloların constraintlerinin import edilip edilmeyeceğini belirtir.
- DESTROY : Tablespace veri dosyalarının verilerini silip üzerine yazılmasını sağlar. İlk değeri N dir.
- ANALYZE : İlk değeri Y dir. Import işleminde SQL ANALYZE ifadesinin kulanım durumunu belitir. SQL ANALYZE export dosyasında bulunur ya da tablolar,indeksler ve kolonlar için istatistik optimizer tarafından yüklenir.

#### 6.4.2.1 Parametre dosyası

Parametre dosyası import parametrelerini içerir. Değişiklik ve tekrar kullanımı kolaylığı için paramere dosyaları tercih edilir. Bu dosyalar tüm text editörlerinden yaratılabilir.

- imp parfile=filename

- imp kullanıcı_adı/şifre parfile=filename

Dosyada kullanılan parametre yazılımları şu şekillerde olabilir:

ANAHTAR_KELİME=değer

ANAHTAR_KELİME = (değer)

ANAHTAR_KELİME = ( değer 1, değer 2, ...)

Dosyaya içersine başına # karakteri koyularak açıklama yazılabilir.

Parametre Dosyası yazılım dizilimi şu şekildedir;

```
imp system/manager file=export.dat  full=Y commit=Y buffer=64000

  FROMUSER=gnl TOUSER=stok ignore=Y rows=Y indexes=Y
```

**Importtan önce tablo yaratımı**

Bir export dosyasından import yapmadan önce kullanıcı tarafından tablolar yaratılabilir. Tablolar aynı yapıyla yaratılmalıdır. Kolonların genişlikleri artırılabilir ya da sıraları değiştirilebilir. Fakat aşağıdaki işlemler yapılmamalıdır.

1. NOT NULL kolonlar eklenemez.

2. Kolonların tipleri birbiriyle uyumsuz veri tipleriyle değiştirilemez. Örneğin LONG tipindeki bir kolon NUMBER olamaz.

3. Referans Kısıtlamalarını Geçersiz kılmak : Import'un normal sırasında referans kısıtlamalarının import'u tüm tabloların import'undan sonradır. Bu sıra bütünlük kısıtlamalarının hataya neden olmasını engeller. Otomatik olarak tüm kısıtlamalar geçersiz kılınır, tablo verileri aktarılır ardından kısıtlamalar yürürlüğe konulur. Fakat varolan bir tabloya import edilirken referans alınan tablo daha import edilmediyse, hata verir. Bu nedenle varolan tablolara importla veri aktarılmadan önce elle tüm kısıtlamalar geçersiz kılınırsa hata mesajı engellenebilir.

İşlem sırası;Import sonrası kısıtlamaların tekrar geçerli olması, büyük tablolar için uzun zaman alabilir. Bu zaman kaybını engellemek için import sırası değiştirilebilir.

Bir export dosyasından bir import yapmak yerine birçok import yapılabilir. İlk olarak referans

alınan tablolar import edilir, sonrasında bunları referans alan tablolar import edilir. bu metod bir tablo kendini referans almıyor ve tablolar arası referans kısır döngüye girmiyorsa işe yarar.

# 7. STOK MODÜLÜ

## 7.1 Modülün Amaçları

- Stoğa giriş ve çıkışların yapılabilmesi,
- Stoğun son durumunun takibi ve bu doğrultuda raporlama,
- Asgari ve azami tutarlar ışığında stok takibi ve uyarı sistemi,
- Departmanların stok harcamaları ve talep takibi,
- Malların fiyat takibi,
- Fatura takibi ve muhasebe ile entegrasyonu,
- Bütçe ile karşılaştırmalı takip.

## 7.2 Stokdaki Temel Kavramlar Ve İşlemler

- Stok, mallardan oluşur ve her biri bir kod alarak tanımlanır,
- Stoklar bir havuzda (depoda) toplanır,
- Stok hareketi iki çeşittir.
  - Departmanların talepleri doğrultusunda yapılan iç transferler; İç transferler taleplere bağlı olduğu gibi ilerde devreye girebilecek rutin transferler de olabilir.
  - Stokta mal ihtiyacının doğmasıyla yapılan alımlar.

Alımların iki tetikçisi vardır. Bunlardan biri departman taleplerinin stoktan karşılanamama durumudur. Diğeri ise mal miktarlarının kritik seviyeye yaklaşmasıdır. Tüm alımların nedeni malları kritik seviyenin üzerinde tutmaktır.

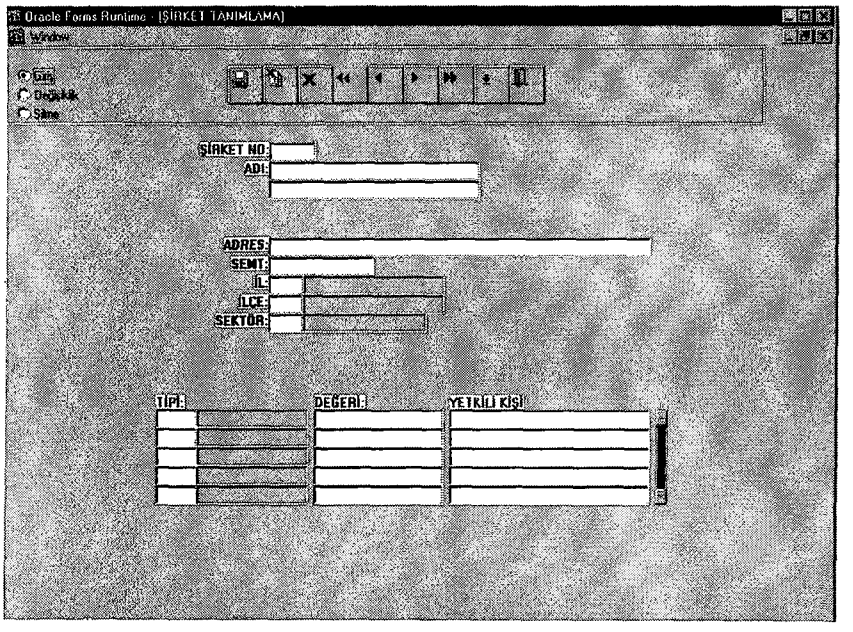
### 7.2.1 Modüldeki programlarda kullanılacak parametreler ve tanımlama ekranları

Operasyonel işlemlerde kullanılacak parametreler şunlardır;

#### 7.2.1.1 Şirket tanımlama

Bu şirketler satın alma departmanının ilişkide bulunacağı tüzel kişilerdir. Tekliflerin

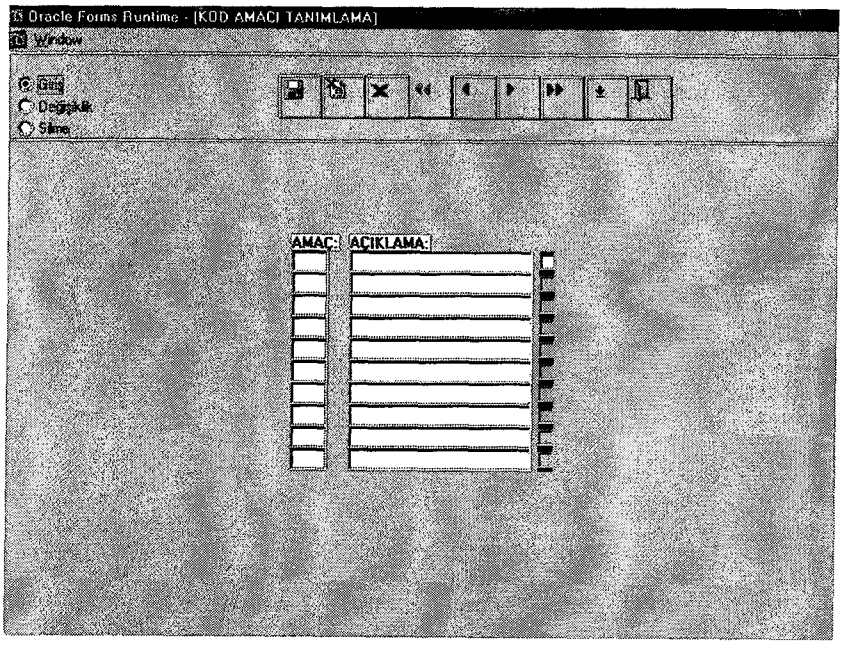
gönderileceği şirketler Şekil 7.1 ekranından tanımlanacaktır.



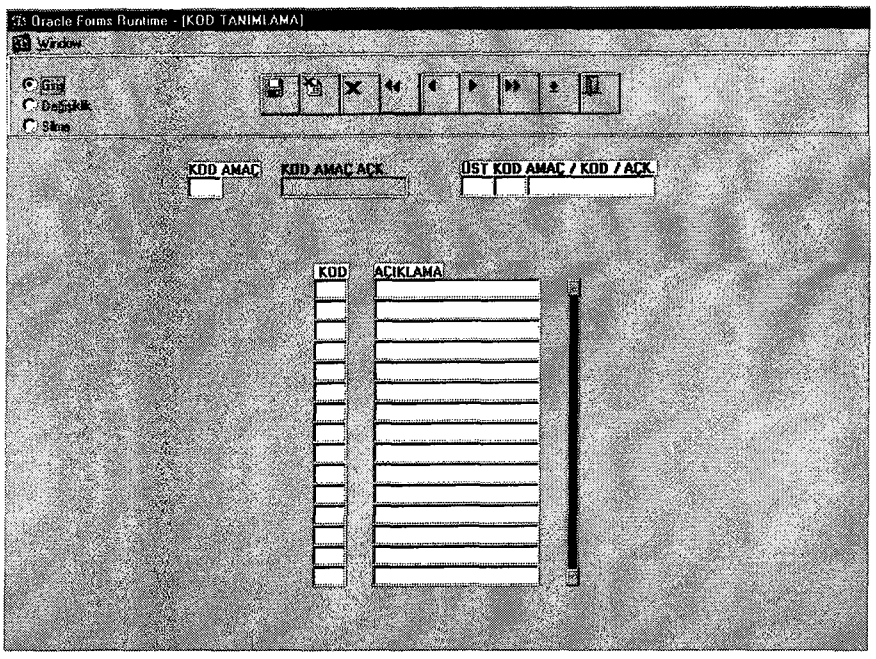

Şekil 7.1 Şirket tanımlama ekranı

### 7.2.1.2 Kod tanımlama

Programda kullanılacak parametreler genelde kod tanımlama programı yardımıyla tanımlanmıştır. Her kodun bir amacı vardır. Kod tanımlamadan önce kodun amacı Şekil 7.2 deki ekrandan tanımlanır.



Şekil 7.2 Kod amacı tanımlama ekranı



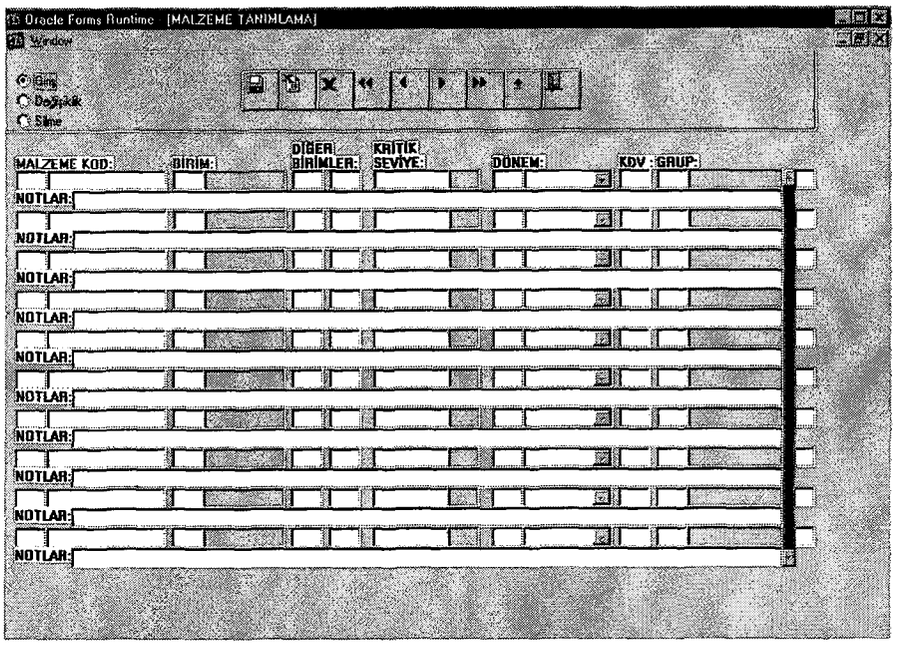

Şekil 7.3 Kod tanımlama ekranı

Kullanılan kodlar;

- İl ve ilçe kodları
- Malzeme grup kodları : Şekil 7.3'den tanımlanacak grup kodları Şekil 7.4 den tanımlanacak olan malzemelerin sınıflandırılmasını sağlar. Bu gruplar bütçe kalemleri olarak atanırsa, bütçe ile kurulacak entegrasyonu kolaylaştıracaktır.
- Birimler
- İletişim kodları
- Departman kodları

### 7.2.1.3 Malzeme kodları tanımlanma

Şekil 7.4'den tanımlanırlar. Her bir malzemenin bir kodu vardır. Bu malların en çok kullanılan birimi ana birimdir ve diğer yardımcı birimler ikinci ve üçüncü birimlerde tanımlanabilir. Kritik seviyesiler ise bu tanımlama ekranından tanımlanır ve aylık,3 aylık,6 aylık ve yıllık olmak üzere dönemleri belirtilir. Ayrıca zaman içersindeki kritik seviye değişimleri güncellenebilir.




Şekil 7.4 Malzeme tanımlama

### 7.2.2 Departmanlardan gelen talepler üzerine izlenen prosedür

Şekil 7.5 departmanların taleplerini girdikleri ekrandır. Bu ekran yetkilendirilmiş personel tarafından kullanılabilir. Talep numarası otomatik dolar, talep tarihi o anki tarihtir. Talep edilen tüm malzemeler miktarlarıyla birlikte belirtilir. Ayrıca malzeme kodu diğer diye tanımlanmış olan bir kod seçilerek ve açıklama alanına bilgisi verilerek talepte bulunulabilir. Böyle talepler satın alma tarafından reddedilebilir ya da gerekli görülürse açıklamadaki malzeme tanımlanır.

Satın alma günün herhangi saatinde tüm talepleri inceler ve stoktaki kritik seviyeleri aşmamak üzere mal transferini gerçekleştirebilir.

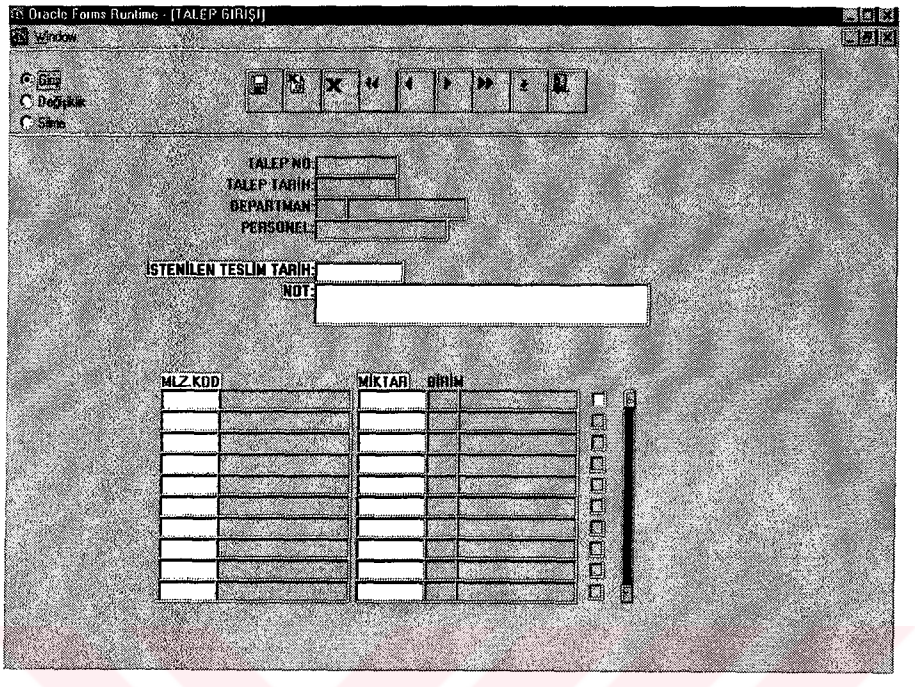
Taleplerin incelenmesi Şekil 7.6'deki ekrandan yapılır. Taleplerin her biri bir durum kod alır. Üç tip kod vardır;

1) Onay - O: Eğer stokta yeterli miktarda mal varsa talep onaylanır ve departmanlara transfer gerçekleştirilir.

2) Bekleme – B: Eğer verilen talep üzerine mallar kritik seviyeye yaklaşıyorsa talep karşılanamaz, bekleme kodu alır.

3) Ret – R: Yukarıda da belirtildiği üzere mal kodların tanımlanmamış bir maldan talep

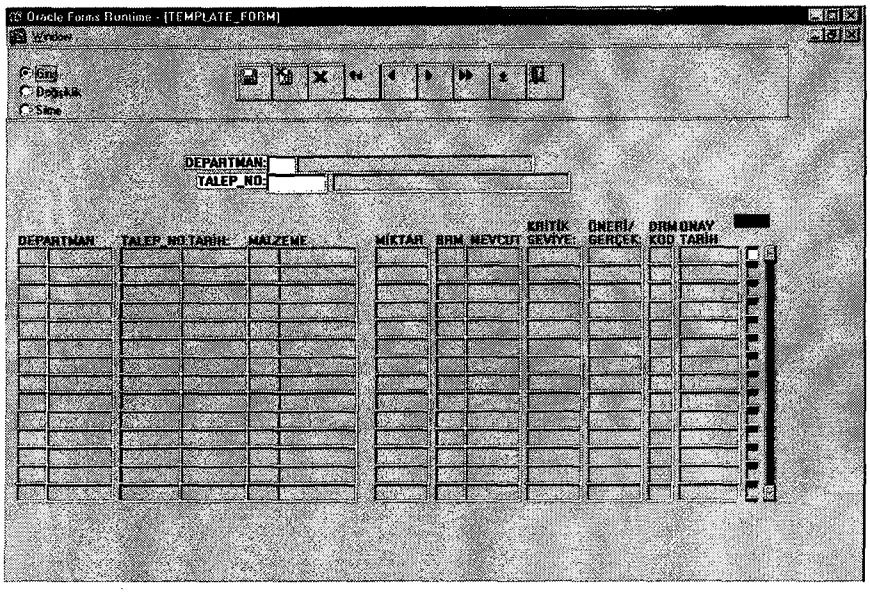
edilmiş ve satın alama tarafından uygun görülmemişse talep reddedilir. Ayrıca verilen talepteki mal veya miktarı o departman için gereksiz görülürse de reddedilebilir.




Şekil 7.5 Talep giriş ekranı

Şekil 7.6'dan tüm talepler verildikleri tarih sırasında departman ve stok kodlarına göre sıralı olarak ekrana gelir. Ekrana gelirken tüm talepler mevcut stok durumuna göre ve kritik seviyeyi aşmamak kaydıyla sırayla doyurulur. Her biri otomatik bir durum kodu alır.Satın alma departmanı yetkilisi tarafından tüm bu talepler incelenir. Örneğin acillik derecesi daha yüksek departmanlar için değişiklikler yapılır. Durum kodları verilerek, son durumları onaylanır.

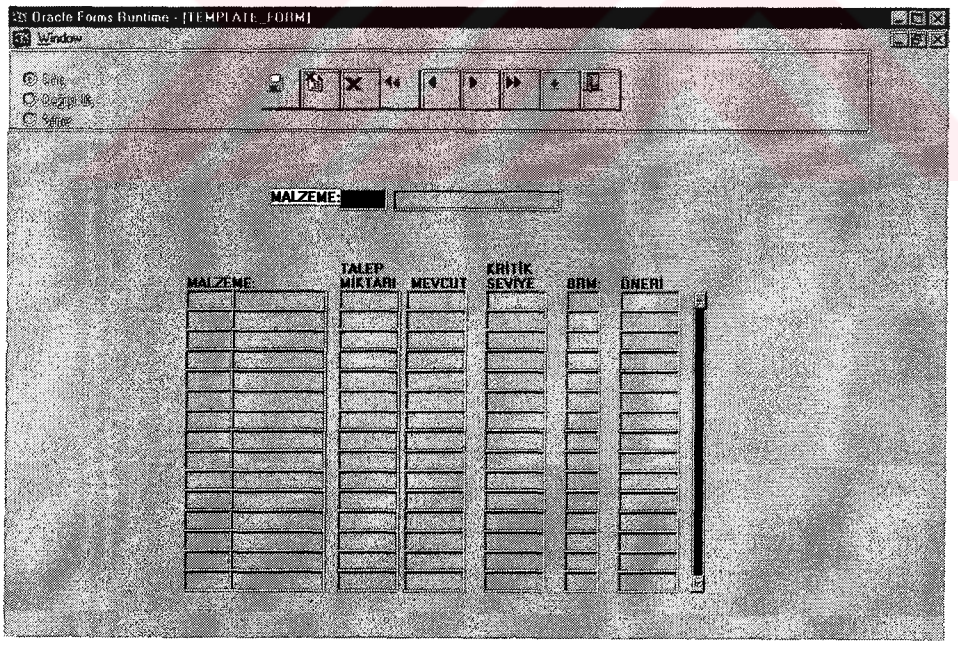
Eğer bir talep onay koduna onay almışsa talep gerçekleşmiş sayılır ve onay tarihi teslim tarihine eşit olur. Böylelikle mal stoktan da çıkmış olur. Bir talebin onay kodu alabilmesi için talep edilen tüm malların karşılanması gerekmektedir.




Şekil 7.6 Talep değerlendirme ekranı

Beklemede olan talepler stoktan karşılanamayanlardır. Bu tür talepler için teklifler hazırlanır.

Şekil 7.7'deki son durum değerlendirme ekranı yardımıyla kritik seviyeye yaklaşmış malzemeler için önerilen miktarlar görüntülenir.

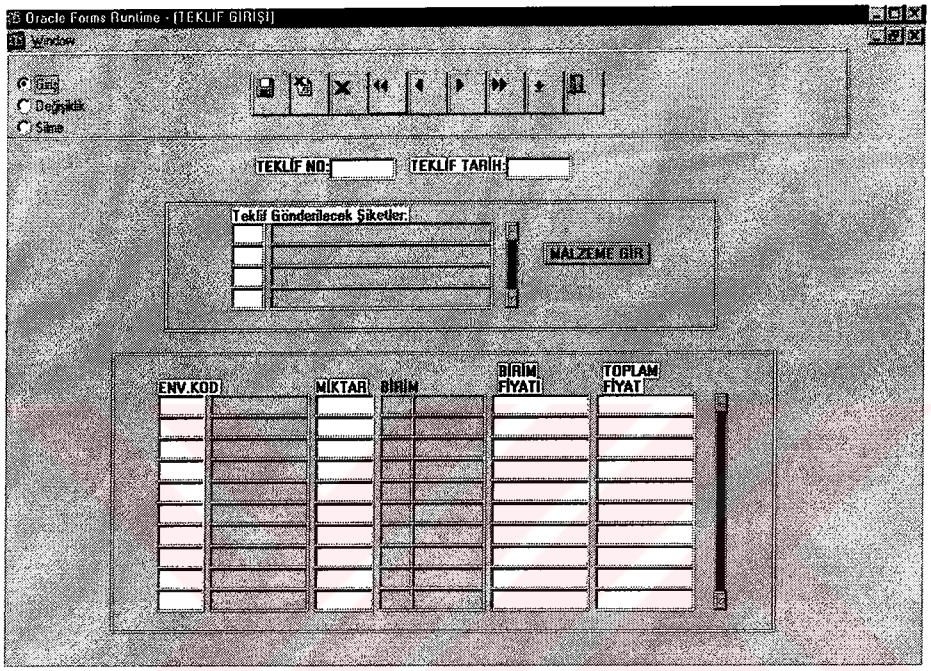



Şekil 7.7 Son durum değerlendirme ekranı

Örneğin talep edilen mal dosya olsun. 20 adet talep edilmiş. Stokta 40 adet var, eğer hepsi

karşılanırsa stokta 20 adet kalacaktır, oysa ki kritik seviyemiz 30'du. Kritik seviyeyi aşmamak için en az alınması gereken adet 10'dur. Bu önerilen bir miktardır, üzerinde mal alımı gerçekleşebilir.

Şekil 7.7 yardımcı ekranıyla belirlenen malzemeler için teklif alınması gerekmektedir. Şekil 7.8 ile teklifler hazırlanır.

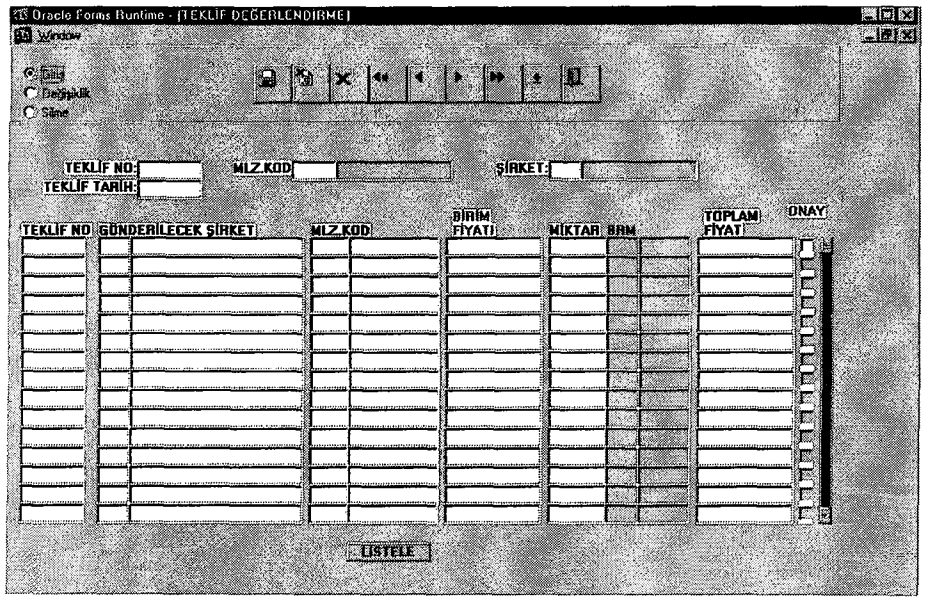



Şekil 7.8 Teklif giriş ekranı

Teklifler ihtiyaç duyulan malzemeler içerirler ve aynı teklif birden çok şirkete gönderilebilir. Tekliflerin sonucunda alınan fiyatlar da bu ekrandan girilir.

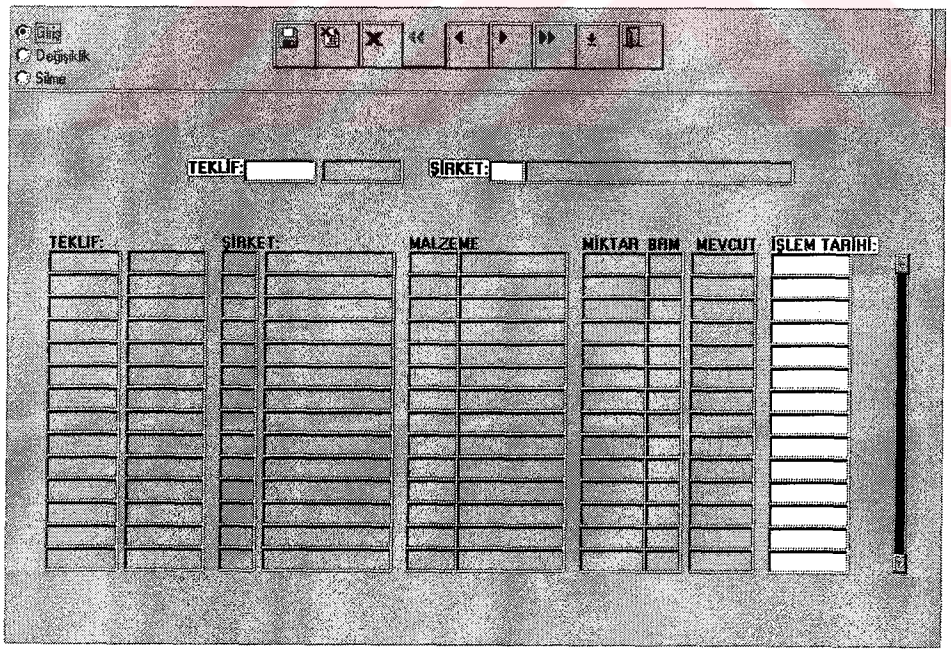
Onayı alan firmaya teklifini referans göstererek bir olur yazısı gönderilir ve diğerlerine de teşekkür yazısı yollanır.

Onaylanan teklife istinaden yapılan alımlar sonucu malların stoğa girişi Şekil 7.9'dan yapılır. Malzeme kodları bazında girişler yapılır.




Şekil 7.9 Teklif değerlendirme ekranı

Girişler yapıldıktan sonra, Şekil 7.6'ya tekrar dönülür. O ana kadar karşılanmayan tüm taleplere (talep kodu = 'B' olan ve henüz durum kodu almayan talepler) otomatik olarak talep tarihi sırasına göre stoktaki mal taleplere dağıtılır. Satın alma yetkilisinin onayı ile taleplerin durumları onaylanır. Durum kodu 'O' yani onay olan malların transferi yapılmış olur.




Şekil 7.10 Malzeme girişi

## 7.3 Tablolar

Stok modülü aşağıdaki tablolardan oluşmaktadır;

- Envanter_table
- Şirket_Table
- Şirket_Irtibat_Table
- Kod_Table
- Kod_Amac_Table
- Sayac_Table
- Message_Table
- Yetki_Table
- Talep_Table
- Talep_Detay_Table
- Teklif_Table
- Teklif_Detay_Table
- Stok_Hareket_Table
- Stok_Durum_Table

Bu tabloların tanımlama metinleri Ek 1'de verilmiştir.

## 7.4 Modüldeki Program Örnekleri

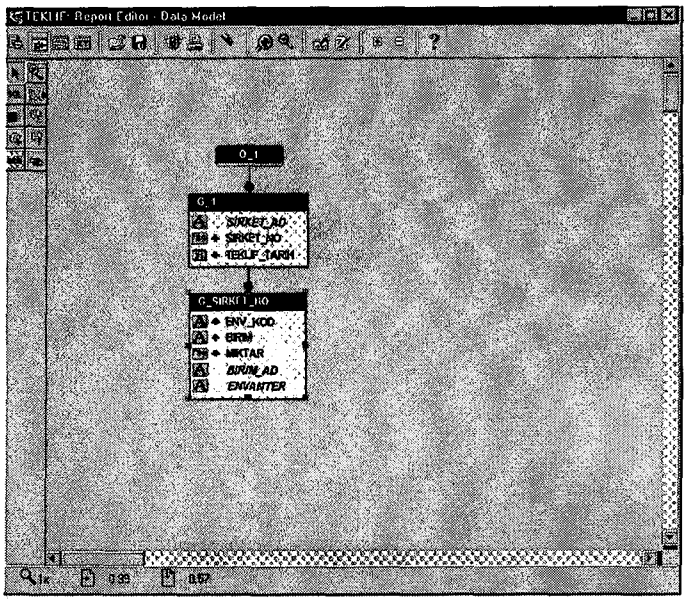
### 7.4.1 Rapor örneği

Raporlar bölüm 5'de anlatılan form builder ile tasarlanmıştır. Örnek olarak Şekil 7.7'deki teklif giriş formundan çağrılan rapor incelenecektir.

Rapor adı TEKLIF'dir. Amacı şirketlerden teklif istenecek malzemelerin listesinin bir üst yazıyla beraber hazırlamaktır.

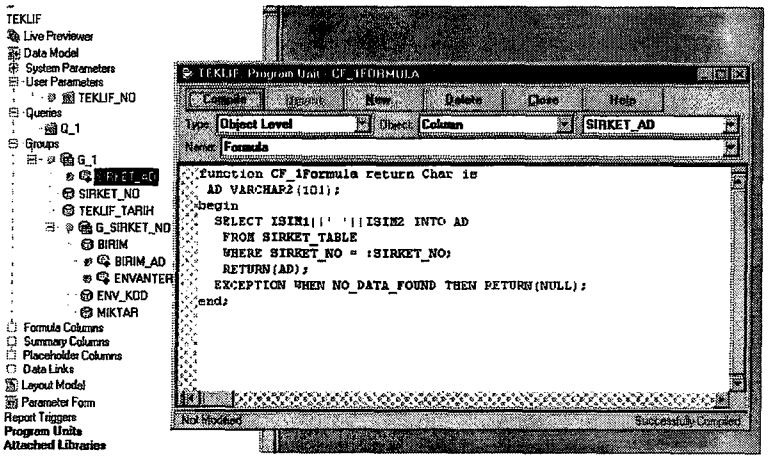
Bölüm 1.3 de tanımlaması verilen TEKLIF_TABLE ve TEKLIF_DETAY_TABLE raporda kullanılmaktadır. Parametresi teklif numarasıdır.

Report builder'da ilk aşama data modeli oluşturmaktır. Eğer wizard yardımıyla rapor oluşturuluyorsa SQL Şekil 7.14'den oluşturulur.



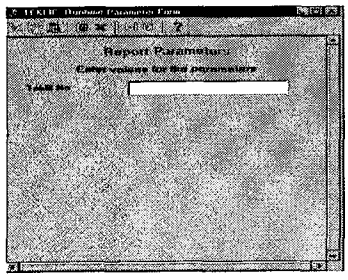

Şekil 7.11 Teklif raporu data modeli

İfade yazıldıktan sonra gerekirse gruplama işlemi yapılır. Rapor teklif parametresi ile çalışmaktadır. Teklifler teklif_table'da yer alırken, detayları telif_detay_table'dadır. Bir teklifin detayı ile kastedilen bilgiler, teklifin gideceği şirketler ve teklifin istendiği malzemelerdir. Şirket ve malzeme bilgileri detaydan alınacaktır. Her bir şirkete bir yazı gideceği için şirket kodunu ayrı bir grup yapmak gerekir. Diğer grup bir şirkete düşen bir veya birden fazla malzeme tipini içerir. Teklif_detay_table'da yalnızca şirket numarası tutulmuştur, şirketin ismini Sirket_table'dan getirilir. Şirket adını almak için bir formula column kullanılır. Bu kolon üzerine bir fonksiyon ile ad bulunur.




Şekil 7.12 Şirket adının fonksiyonla bulunması

Birim ad ve envanter kolonları da aynı yöntemle kod_table ve envanter_table'dan bulunur. Şekil 7.13'deki ekran sayesinde rapor direk report builder tarafından çalıştırılırsa parametre alınır.

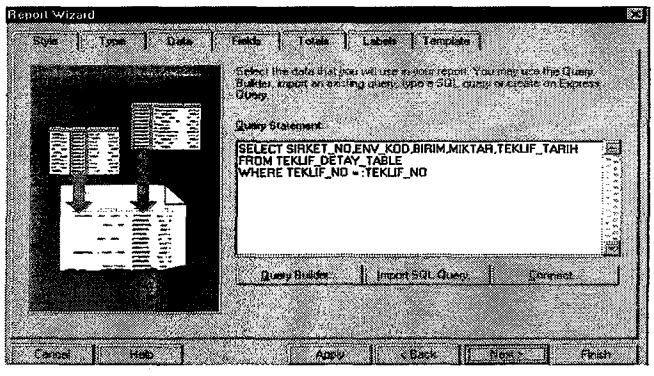


Şekil 7.13 Rapor parametre ekranı

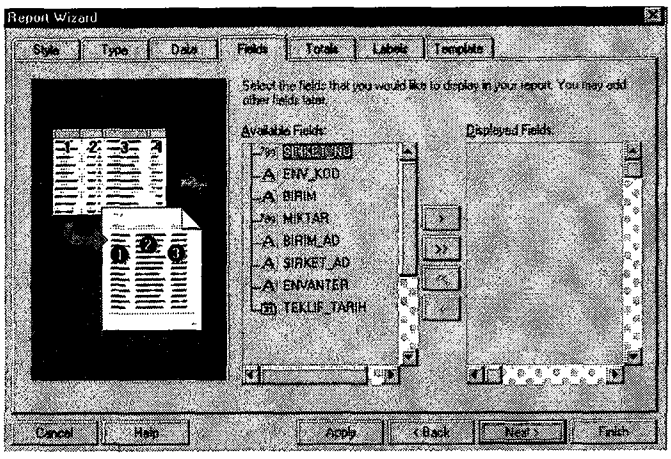
Birçok rapor tipi vardır. Bunlar:

- Tabular
- Form-like
- Mailing label
- Form letter
- Group left
- Group above
- Matrix
- Matrix with group

Bu örnekte Group left tipi kullanılacaktır. Master detail raporlada kullanılan formattır. Wizard yardımıyla rapor oluşturulabileceği gibi elle de oluşturulabilir. Data modelde SQL ifadesi yazılabileceği gibi Şekil 7.14'de gösterilen wizard sayesinde de yazılabilir.



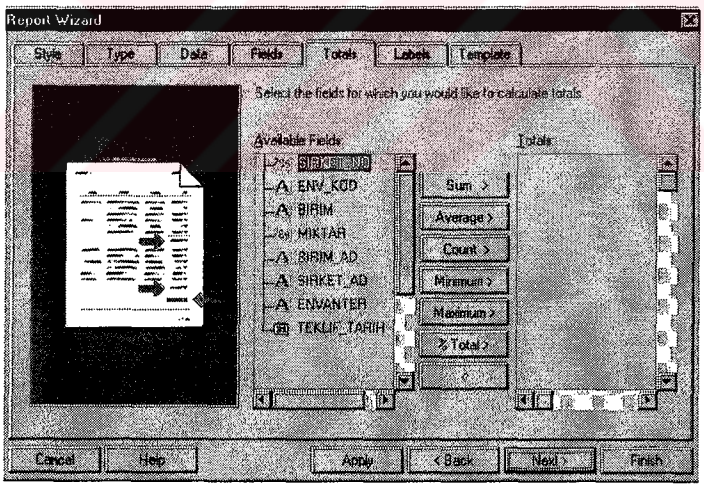

Şekil 7.14 Raporun SQL ifadesi




Şekil 7.15 Rapor kolonlarının seçimi

Layout elle tasarlanırken repeating frame'ler araçlar yardıyla çizilir. Frame bir grubu gösterir. İçindeki alanlar gruba ait olamalıdır. Wizard yardımıyla kolonlar Şekil 7.15'den seçilir. Kolonlara summary kolonlar eklenmesi elle data model bölümünden yapılır, wizar yardımıyla kolon tanımlama Şekil 7.16'daki ekrandan yapılmaktadır.

Rapor tasarım Label sekmesinde kolon başlıkları verilerek ve template rapor tipi seçildikten sonra son bulur.



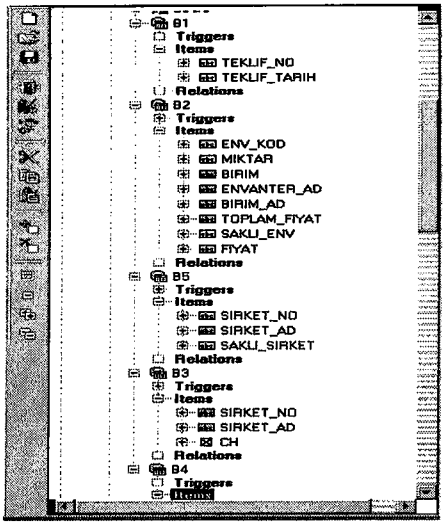

Şekil 7.16 Özet kolonlarının oluşması

### 7.4.2 Form örneği

Formlar bölüm 5'de anlatılan form builder ile tasarlanmıştır. Örnek olarak Şekil 7.7'de gösterilen teklif giriş formu incelenecektir. Program otomatik teklif numarası alarak teklik

bilgilerini TEKLIF_TABLE ve TEKLIF_DETAY_TABLE'a atmaktadır.

Form PAGE_1, PAGE_2 isimli iki kanvas, beş blok ve iki pencereden oluşmaktadır. Şekil 7.17'de gösterildiği gibi blokların altın birçok alan vardır.

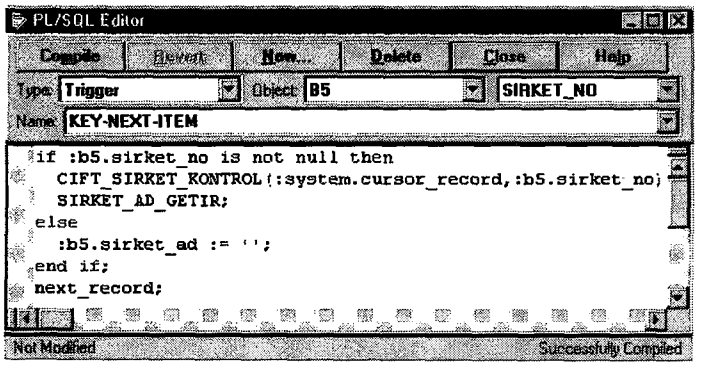



Şekil 7.17 Teklif formunun blokları ve alanları

Ekrana ilk girişte sayac_table'dan bir teklif no alınmaktadır. Sayac_no_al veritabanında saklanan bir fonksiyondur. Bu sayede diğer formlarda da kullanılabilir.

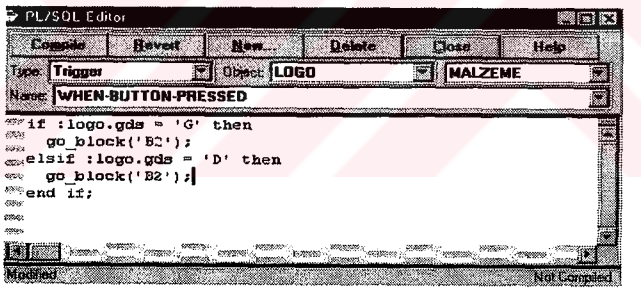
```
FUNCTION SAYAC_NO_AL (saytip in number)
            RETURN NUMBER IS
 sayno number;
BEGIN
BEGIN
   select sayac_no into sayno
    from sayac_table
    where sayac_tip = saytip;
exception when no_data_found then null;
END;
update sayac_table
 set sayac_no = sayno + 1
 where sayac_tip = saytip;
return(sayno+1);
END;
```

Ekrandaki alanlar üzerinde triggerlar sayesinde alan düzeyinde kontrol yapılabilmektedir. Bunlardan biri şirket no alanıdır.



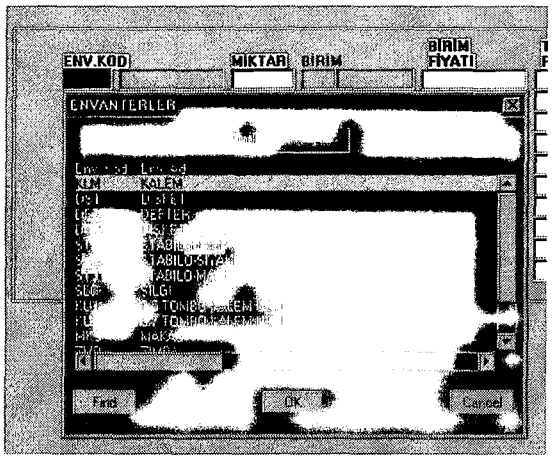

Şekil 7.18 Key-next-item tetiği

Üzerinde Şekil 7.18'deki KEY-NEXT-ITEM'ı vardır. Bu PL/SQL bloğu çift_sirket_konrol prosedürünü çağırmaktadır. Bu sayede aynı şirket iki kere girilemeyecektir. Eğer çift kayıt değilse bir alttaki satırdaki sirket_ad_getir prosedürü ile ekrandaki şirket adı dolar. Teklif istenecek şirketler girildikten sonra malzeme gir düğmesine basılarak malzeme veri bloğundan malzemeler seçilir. Bu sırada düğme üzerindeki Şekil 7.19'daki tetkik çalışır.



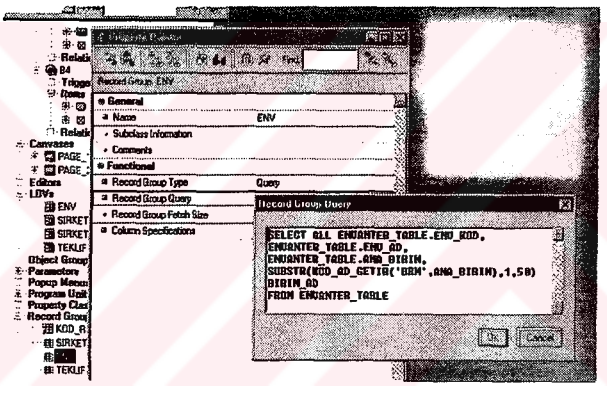

Şekil 7.19 When-button-press tetik örneği

Alanların üzerinde List of Values denilen bir sql ifadesinden oluşan yardımcı ekranlar vardır. Envanter(malzeme) kodunda Şekil 7.20'deki list of values vardır.



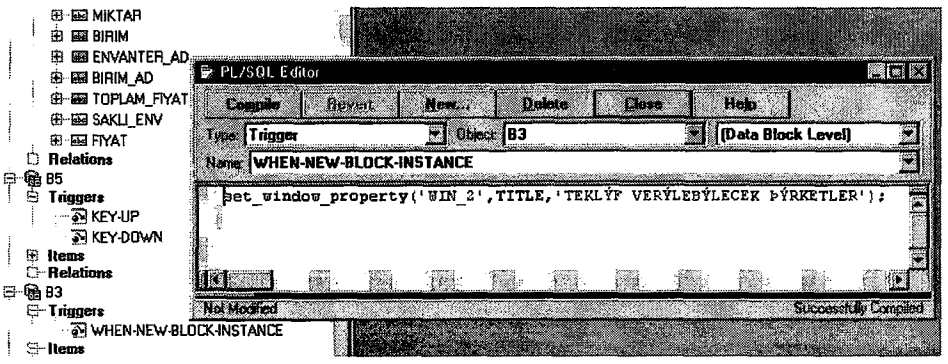

Şekil 7.20 Listeleme ekranı

LOV'lar değerlerini RECORDS GROUPS dan alırlar. Envanterin record groubu Şekil 7.21'deki gibidir.




Şekil 7.21 Kayıt grubu örneği

Ekran alanlar arasındaki hareket esnasında blok düzeyinde tetikler yazılabilir. Örneğin b3 bloğuna girildiğinde çalışacak WHEN-NEW-BLOCK-INSTANCE tetiği Şekil 7.22'de yer almaktadır. B3 bloğu şirketleri gösteren kanvasın üzerine yer alır. Bu blok farklı bir kanvasta yer aldığından penceresi de değişebilir. Set_Window_Property built-in sayesinde pencerenin başlığı değiştirilir.




Şekil 7.22 Blok düzeyinde tetik örneği

Ekrandaki alanlar doldurulurken ekranın üzerindeki düğmeler de kullanılır. Bunlar işlemlerin kaydedilmesine, kaydın silinmesine, tablonun silinmesine, üzerindeki alanın LOV'unun gösterilmesine, alanlar ve bloklar arası geçişe ve programdan çıkışa yararlar. Kayıt sil düğmesinin üzerindeki tetik aşağıdaki prosedürü çağırmaktadır. Bu prosedür formun program ünitesi bölümde saklanır. Bir PL/SQL bloğunu içeren prosedür imlecin bulunduğu bloğa göre farklı tablolardan kayıt silmektedir. Bu örnekte :system.cursor_block built-in kullanılmıştır. İmlecin bulunduğu bloğu gösteren bu built-in gibi :system ile başlayan çok sayıda built-in vardır. Karakter değerler döndüren bu built-in ler forma özgü bir fonksiyonlardır.

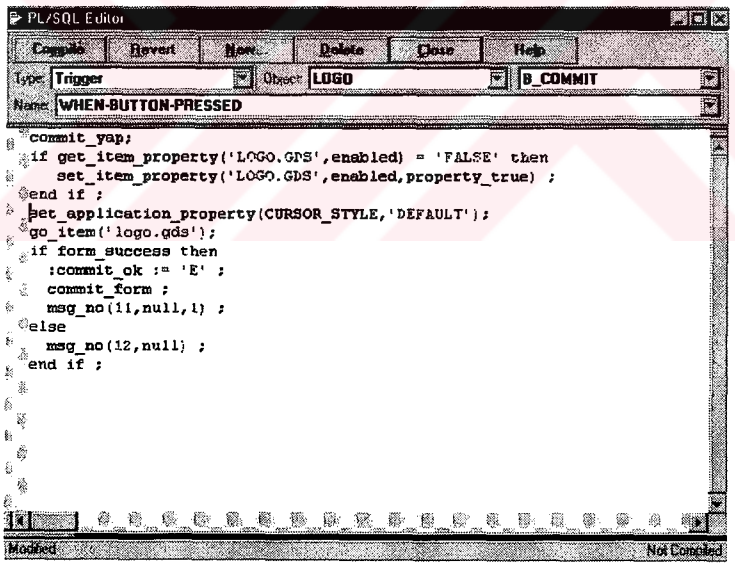
```
PROCEDURE kayit_sil IS
var number(1);
BEGIN
 if :system.cursor_block = 'B1' then
   delete teklif_table
   where teklif_no = :b1.teklif_no;
   elsif :system.cursor_block = 'B5' then
 delete teklif_detay_table
   where teklif_no = :b1.teklif_no and
       sirket_no = :b5.sirket_no;
 elsif :system.cursor_block = 'B2' then
 delete teklif_detay_table
   where teklif_no = :b1.teklif_no and
       env_kod = :b2.env_kod;
 end if;
END;
```

Programa değişiklik modu ile girildiğinde teklif numarasından TEKLIF_TABLE ve TEKLIF_DETAY_TABLE'dan ilgili kayıtllar getirilir. Bu işlem bir cursor ve fetch yardımıyla yapılmaktadır. Kayıt getir prosedürü ile iki tane cursor tanımlanmıştır. Girilen teklif numarasından kapalı bir cursor ile teklif tarihi getirilmiştir. Açık imleçler c1 ve c2 open ile açışmış, close ile kapatılmıştır. Bir döngünün içinde fetch ile alanlara cursor değerleri atanmıştır. Döngüden c1%notfound true döndüğü zaman çıkılır.

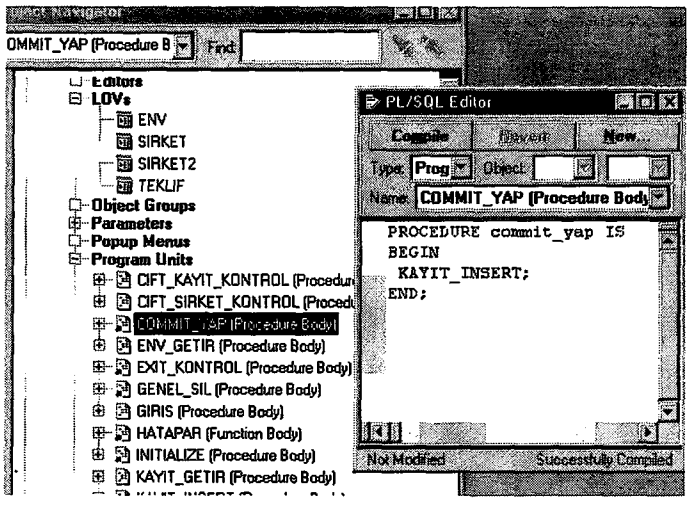
```
PROCEDURE KAYIT_GETIR IS
cursor c1 is
 select distinct env_kod,birim,miktar,fiyat
  from teklif_detay_table
  where teklif_no = :b1.teklif_no ;
cursor c2 is
  select distinct sirket_no
  from teklif_detay_table
  where teklif_no = :b1.teklif_no ;
say number(2);
BEGIN
select teklif_tarih into :b1.teklif_tarih
 from teklif_table
 where teklif_no = :b1.teklif_no ;
 go_block('b2');
 first_record;
 open c1;
 loop
  fetch c1 into :b2.env_kod,:b2.birim,:b2.miktar,:b2.fiyat;
  :b2.sakli_env := :b2.env_kod;
  :b2.toplam_fiyat := :b2.fiyat * :b2.miktar;
  exit when c1%notfound;
  env_getir;
  next_record;
end loop;
close c1;
```

```
---------------------------------------------------------------------------

go_block('B5');

first_record;

open c2;

loop

fetch c2 into :b5.sirket_no;

exit when c2%notfound;

sirket_ad_getir;

next_record;

end loop;

close c2;

END;
```

İşlemler tamamlandıktan sonra işlemleri kaydetmek için kayıt düğmesine basılır. Bu hareket Şekil 7.23'deki tetiği çalıştırır. Tetikten commit_yap prosedürü çağrılmaktadır. Bu prosedür Şekil 7.24'de gösterilen program ünitesi bölümde saklanır.




Şekil 7.23 Kaydetme düğmesi tetiği




Şekil 7.24 Commit_yap program ünitesi

**KAYNAKLAR**

Codd, A., (1970), Relational Model Of Data For Large Shared Data Banks, Communications of the ACM.

Date, C.J., (1981), An Introduction to Database Systems, Addison-Wesley Publishing Company.

Date, C.J., (1990), An Introduction to Database Systems Volume I, Addison-Wesley Publishing Company.

Elmasri R. ve Navathe S.B., (1994), Fundamentals of Database Systems, The Benjamin/Cummings Publishing Company.

Oracle, (1999a), Enterprise DBA Architecture and Administration Volume1-2 Student Guide, Oracle Corporation, Ireland.

Oracle, (1999b), Oracle 8i Release 8.1.5 Concepts Volume 1-2, Oracle Corporation, Ireland.

Oracle, (1999c), Oracle Report Developer: Build Internet Reports Volume 1-2 Student Guide, Oracle Corporation, Ireland.

Oracle, (1999d), Oracle 8i Release 8.1.5 SQL Reference, Volume 1-2, Oracle Corporation, Ireland.

Oracle, (1999e), SQL*Plus Release 8.1.5 User's Guide and Reference, Oracle Corporation, Ireland.

Oracle, (1999f), PL/SQL 8i Release 8.1.5 User's Guide and Reference, Oracle Corporation, Ireland.

Oracle, (1999g), Oracle 8i Release 8.1.5 Utilities, Oracle Corporation, Ireland.

**EKLER**

## Ek 1 Stok modülünde kullanılan tabloların tanımlama metinleri

Modülü oluşturan tablolar ve yapıları;

```
drop table envanter_table cascade constraints;
create table envanter_table(
    env_kod varchar2(3) constraint nn_stok_tab_stk_kod not null,
    env_ad varchar2(100) constraint nn_stok_tab_ad not null,
    aciklama varchar2(200),
    sektor_kod varchar2(3) ,
    ana_birim varchar2(3) ,
    birim2 varchar2(3) ,
    birim3 varchar2(3) ,
    grup_kod varchar2(3) ,
    kritik_seviye number,
    kritik_seviye_birim varchar2(3),
    donem number(3),
    periyod varchar2(1), -- yil,ay,gun
    kdv_oran number,
    tarih date default sysdate,
    op_id varchar2(12) default substr(user,1,12))
    tablespace stk
    storage (initial 500k next 50k pctincrease 0);
drop public synonym envanter_table;
create public synonym envanter_table for stk.envanter_table;
grant all on stk.envanter_table to public;
alter  table envanter_table add constraint pk_stk_kod_tab_stkod
primary key  (env_kod)
using index pctfree 15
tablespace stk_ndx
storage (initial 100k next 10k pctincrease 0);
drop table sirket_table cascade constraints;
create table sirket_table(
    sirket_kod number(3) constraint nn_sirket_tab_sk not null,
    isim1 varchar2(50) constraint nn_sirket_tab_isi1 not null,
    isim2 varchar2(50) constraint nn_sirket_tab_isi2 not null,
    sektor_kod varchar2(3) ,
    adres varchar2(50) ,
    semt varchar2(25),
    ilce_kod varchar2(3),
```

```
    il_kod varchar2(3),
    tarih date default sysdate,
    op_id varchar2(12) default substr(user,1,12))
    tablespace stk
    storage (initial 100k next 10k pctincrease 0);
drop public synonym sirket_table;
create public synonym sirket_table for stk.sirket_table;
grant all on stk.sirket_table to public;
comment on table  stk.sirket_table is 'ÞÝRKET TABLE';
comment on column stk.sirket_table.sirket_kod is 'ÞÝRKET KODU';
comment on column stk.sirket_table.isim1 is 'ÝSÝM1';
comment on column stk.sirket_table.isim2 is 'ÝSÝM2';
comment on column stk.sirket_table.sektor_kod is 'SEKTÖR';
comment on column stk.sirket_table.adres is 'ADRES';
comment on column stk.sirket_table.semt is 'SEMT';
comment on column gnl.sirket_table.ilce_kod is 'ÝLÇE';
comment on column gnl.sirket_table.il_kod is 'ÝL';
comment on column gnl.sirket_table.op_id is 'KULLANICI ADI';
comment on column gnl.sirket_table.tarih is 'TARÝH';
alter  table sirket_table add constraint pk_sirket_tab_sk
primary key  (sirket_kod)
using index pctfree 15
tablespace stk_ndx
storage (initial 10k next 1k pctincrease 0);
drop table sirket_irtibat_table cascade constraints;
create table sirket_irtibat_table(
    sirket_kod number(3) constraint nn_sirket_irt_tab_sk not null,
    irtibat_tip varchar2(3) constraint nn_sirket_irt_tab_irttip not null,
    sira_no number(2) constraint nn_sirket_irt_tab_irttip not null,
    irtibat_no varchar2(50) constraint nn_sirket_irt_tab_irtno not null,
    ilgili_ad  varchar2(50) ,
    ilgili_soyad  varchar2(50) ,
    ilgili_unvan  varchar2(50) ,
    tarih date default sysdate,
    op_id varchar2(12) default substr(user,1,12))
    tablespace stk
    storage (initial 100k next 10k pctincrease 0);
drop public synonym sirket_irtibat_table;
create public synonym sirket_irtibat_table for stk.sirket_irtibat_table;
```

```
 grant all on stk.sirket_irtibat_table to public;
alter  table sirket_irtibat_table add constraint pk_sirket_irt_tab_sk_irttip_sno
 primary key  (sirket_kod,sira_no)
 using index pctfree 15
 tablespace stk_ndx
 storage (initial 10k next 1k pctincrease 0);
drop table kod_amac_table cascade constraints;
create table kod_amac_table(
     kod_amac varchar(3) constraint nn_kod_amac_tab_kod_amac not null,
     aciklama varchar2(200),
     kullanildi varchar2(1),
     tarih date default sysdate,
     op_id varchar2(12) default substr(user,1,12))
     tablespace stk
     storage (initial 100k next 10k pctincrease 0);
 drop public synonym kod_amac_table;
 create public synonym kod_amac_table for stk.kod_amac_table;
 grant all on stk.kod_amac_table to public;
 alter  table kod_amac_table add constraint pk_kod_amac_tab_kdamc
 primary key  (kod_amac)
 using index pctfree 15
 tablespace stk_ndx
 storage (initial 10k next 1k pctincrease 0);
drop table kod_table cascade constraints;
create table kod_table(
     kod_amac varchar(3) constraint nn_kod_tab_kod_amac not null,
     kod varchar2(3) ,
     aciklama varchar2(200),
     ust_kod_amac varchar2(3),
     ust_kod varchar2(3),
     ust_aciklama varchar2(200),
     tarih date default sysdate,
     op_id varchar2(12) default substr(user,1,12))
     tablespace stk
     storage (initial 250k next 25k pctincrease 0);
drop public synonym kod_table;
create public synonym kod_table for stk.kod_table;
grant all on stk.kod_table to public;
alter  table kod_table add constraint uq_kod_tab_kod
```

```
 unique (kod_amac,kod,ust_kod_amac,ust_kod)
 using index pctfree 15
 tablespace stk_ndx
 storage (initial 10k next 1k pctincrease 0);
drop sayac_table cascade constraints;
create table sayac_table(
    sayac_tip number constraint nn_sayac_tab_sayac_tip not null,
    sayac_no number constraint nn_sayac_tab_sayac_no not null,
    aciklama varchar2(200),
    tarih date default sysdate,
    op_id varchar2(12) default substr(user,1,12))
    tablespace stk
    storage (initial 100k next 10k pctincrease 0);
 drop public synonym sayac_tabl;
 create public synonym kod_table for stk.sayac_tabl;
 grant all on stk.sayac_table to public;
 alter  table sayac_table add constraint pk_sayac_table_sayactip
 primary key  (sayac_tip)
 using index pctfree 15
 tablespace stk_ndx
 storage (initial 10k next 1k pctincrease 0);
drop table message_table cascade constraints;
create table message_table(
    code varchar2(10) ,
    cause varchar2(2000),
    action varchar2(2000),
    text varchar2(2000),
    texttr varchar2(2000))
    tablespace stk
    storage (initial 250k next 25k pctincrease 0);

drop public synonym message_table ;
create public synonym message_table for stk.message_table ;
grant all on stk.message_table  to public;
alter  table message_table add constraint pk_message_table_code
primary key  (code)
using index pctfree 15
tablespace stk_ndx
storage (initial 10k next 1k pctincrease 0);
```

```
drop table yetki_table cascade constraints;
create table yetki_table(
    program_ad varchar2(15) ,
    kullanici varchar2(15),
    dept_kod varchar2(3),
    folder varchar2(250))
    tablespace stk
    storage (initial 100k next 10k pctincrease 0);
drop public synonym yetki_table ;
create public synonym yetki_table for stk.yetki_table ;
grant all on stk.yetki_table  to public;
alter  table yetki_table add constraint pk_yetki_table_code
primary key  (program_ad,kullanici)
using index pctfree 15
tablespace stk_ndx
storage (initial 10k next 1k pctincrease 0);

drop table talep_table cascade constraints;
create table talep_table(
    talep_no number constraint nn_talep_tab_tlpno not null,
    talep_tarih date constraint nn_talep_tab_tlp_tar not null,
    talep_durum_kod varchar2(3),
    departman_kod varchar2(3),
    istenilen_teslim_tarih date,
    aciklama varchar2(200),
    tarih date default sysdate,
    op_id varchar2(12) default substr(user,1,12))
    tablespace stk
    storage (initial 100k next 10k pctincrease 0);
drop public synonym talep_table;
create public synonym talep_table for stk.talep_table;
grant all on stk.talep_table to public;
alter  table talep_table add constraint pk_talep_tab_tlpno
primary key  (talep_no)
using index pctfree 15
tablespace stk_ndx
storage (initial 100k next 10k pctincrease 0);
drop table talep_detay_table cascade constraints;
create table talep_detay_table(
```

```
    talep_no number constraint nn_tlp_dty_tab_tlpno not null,
    env_kod varchar2(3) constraint nn_tlp_dty_tab_envkd not null,
    birim varchar2(3) ,
    miktar number,
    talep_durum_kod varchar2(3),
    tarih date default sysdate,
    op_id varchar2(12) default substr(user,1,12))
    tablespace stk
    storage (initial 100k next 10k pctincrease 0);
drop public synonym talep_detay_table;
create public synonym talep_detay_table for stk.talep_detay_table;
grant all on stk.talep_detay_table to public;
alter  table talep_detay_table add constraint pk_talep_dty_tab_tlpno_envkd
primary key  (talep_no,env_kod)
using index pctfree 15
tablespace stk_ndx
storage (initial 1000k next 100k pctincrease 0);
drop table stok_hareket_table cascade constraints;
create table stok_hareket_table(
    hareket_no number constraint nn_stok_har_tab_harno not null,
    referans_no number constraint nn_stok_har_tab_refno not null,
    referans_tarih date constraint nn_stok_har_tab_reftar not null,
    env_kod varchar2(3) constraint nn_stok_har_tab_envd not null,
    islem_tip varchar2(3) constraint nn_stok_har_tab_isltip not null,
    islem_tarih date constraint nn_stok_har_tab_isltar not null,
    sirket_no number(10),
    departman_kod varchar2(3),
    referans_miktar number,
    islem_miktar number constraint nn_stok_har_tab_mkr not null,
    mevcut number constraint nn_stok_har_tab_mvcu not null,
    son_durum_miktar number constraint nn_stok_har_tab_sonmkr not null,
    tarih date default sysdate,
    op_id varchar2(12) default substr(user,1,12))
    tablespace stk
    storage (initial 1000k next 100k pctincrease 0);
drop public synonym stok_hareket_table;
create public synonym stok_hareket_table for stk.stok_hareket_table;
grant all on stk.stok_hareket_table to public;
alter  table stok_hareket_table add constraint pk_stk_har_harnorefnoenv
```

```
 primary key  (hareket_no,referans_no,env_kod)
 using index pctfree 15
 tablespace stk_ndx
 storage (initial 10k next 1k pctincrease 0);
drop table stok_durum_table cascade constraints;
create table stok_durum_table(
    hareket_no number constraint nn_stok_dur_tab_harno not null,
    env_kod varchar2(3) constraint nn_stok_dur_tab_envd not null,
    islem_tip varchar2(3) constraint nn_stok_dur_isltip not null,
    islem_tarih date constraint nn_stok_dur_isltar not null,
    mevcut number constraint nn_stok_dur_tab_mvcu not null,
    tarih date default sysdate,
    op_id varchar2(12) default substr(user,1,12))
    tablespace stk
    storage (initial 1000k next 100k pctincrease 0);
drop public synonym stok_durum_table;
create public synonym stok_durum_table for stk.stok_durum_table;
grant all on stk.stok_durum_table to public;
alter  table stok_durum_table add constraint pk_stok_hsr_tab_harno
primary key  (hareket_no)
using index pctfree 15
tablespace stk_ndx
storage (initial 10k next 1k pctincrease 0);
drop table teklif_table cascade constraints;
create table teklif_table(
    teklif_no number constraint nn_teklif_tab_tklfno not null,
    teklif_tarih date constraint nn_teklif_tab_tklftar not null,
    teklif_durum_kod varchar2(3) ,
    tarih date default sysdate,
    op_id varchar2(12) default substr(user,1,12))
    tablespace stk
    storage (initial 100k next 10k pctincrease 0);
drop public synonym teklif_table;
create public synonym teklif_table for stk.teklif_table;
grant all on stk.teklif_table to public;
alter  table teklif_table add constraint pk_teklif_tab_tek_refno_tekno
primary key  (teklif_no)
using index pctfree 15
tablespace stk_ndx
```

```
 storage (initial 10k next 1k pctincrease 0);
drop table teklif_detay_table cascade constraints;
create table teklif_detay_table(
    hareket_no number(3) constraint nn_teklif_dty_tab_harno not null,
    teklif_no number constraint nn_teklif_dty_tab_tklfno not null,
    sirket_no number constraint nn_teklif_dty_tab_sirkod not null,
    env_kod varchar2(3) constraint nn_teklif_dty_tab_envkod not null,
    teklif_durum_kod varchar2(3) ,
    teklif_tarih date constraint nn_teklif_dty_tab_tklftar not null,
    birim varchar2(3) constraint nn_teklif_dty_tab_brm not null,
    miktar number constraint nn_teklif_dty_tab_mkr not null,
    fiyat number,
    tarih date default sysdate,
    op_id varchar2(12) default substr(user,1,12))
    tablespace stk
    storage (initial 100k next 10k pctincrease 0);
drop public synonym teklif_detay_table;
create public synonym teklif_detay_table for stk.teklif_detay_table;
grant all on stk.teklif_detay_table to public;
alter  table teklif_detay_table add constraint pk_teklif_dty_tab_teksirenv
primary key  (hareket_no,teklif_no,sirket_no,env_kod)
using index pctfree 15
tablespace stk_ndx  storage (initial 10k next 1k pctincrease 0);
```

## Ek 2 Sabit formatlı verinin veritabanına aktarımı sonucu oluşan log dosyası

LOG FILE

SQL*Loader: Release 8.0.5.0.0 - Production on Wed Dec 22 10:41:8 1999



Control File: T:/CTL/pfmes_ctl.txt

Data File: T:/Data/pfmeslk.txt

Bad File: t:\bad\hayat.bad

Discard File: t:\dis\hayat.dis

(Allow all discards)

Number to load: ALL

Number to skip: 0

Errors allowed: 50

Bind array: 1 rows, maximum of 65536 bytes

Continuation: none specified

Path used: Conventional

Table PFMESLK, loaded from every logical record.

Insert option in effect for this table: REPLACE

| Column Name | Position | Len | Term | Encl | Datatype |
|---|---|---|---|---|---|
| MESNO | 3:4 | 2 | | | CHARACTER |
| KZSNF | 6:6 | 1 | | | CHARACTER |
| MSLK | 7:21 | 15 | | | CHARACTER |
| GRRSKT | 24:25 | 2 | | | CHARACTER |

Table PFMESLK:

40 Rows successfully loaded.

0 Rows not loaded due to data errors.

0 Rows not loaded because all WHEN clauses were failed.

0 Rows not loaded because all fields were null.

Space allocated for bind array: 28 bytes(1 rows)

Space allocated for memory besides bind array: 0 bytes

Total logical records skipped: 0

Total logical records read: 40

Total logical records rejected: 0

Total logical records discarded: 0

Run began on Wed Dec 22 10:41:08 1999

Run ended on Wed Dec 22 10:41:08 1999

Elapsed time was: 00:00:00.24

CPU time was: 00:00:00.05

## Ek 3 Değişken uzunlukta verinin veritabanına aktarımı sonucu oluşan log dosyası

SQL*Loader: Release 8.0.5.0.0 - Production on Fri Jan 7 9:15:30 2000



Control File: T:/CTL/subacnpf1_ctl.txt

Bad File: T:/CTL/subacnpf1.bad

Discard File: none specified

(Allow all discards)

Number to load: ALL

Number to skip: 0

Errors allowed: 50

Bind array: 64 rows, maximum of 65536 bytes

Continuation: none specified

Path used: Conventional

Table SUBACNPF1, loaded from every logical record.

Insert option in effect for this table: INSERT

| Column Name | Position | Len | Term | Encl | Datatype |
|---|---|---|---|---|---|
| BNKKD | FIRST | * | , | O(") | CHARACTER |
| SBKOD | NEXT | * | , | O(") | CHARACTER |
| ACENT | NEXT | * | , | O(") | CHARACTER |

Table SUBACNPF1:

75 Rows successfully loaded.

0 Rows not loaded due to data errors.

0 Rows not loaded because all WHEN clauses were failed.

0 Rows not loaded because all fields were null.

Space allocated for bind array: 65016 bytes(84 rows)

Space allocated for memory besides bind array: 0 bytes

Total logical records skipped: 0

Total logical records read: 75

Total logical records rejected: 0

Total logical records discarded: 0

Run began on Fri Jan 07 09:15:30 2000

Run ended on Fri Jan 07 09:15:30 2000

Elapsed time was: 00:00:00.16

CPU time was: 00:00:00.05

## Ek 4 Belirli aralıktaki verilerin veritabanına yüklenmesi sonucu oluşan log dosyası

SQL*Loader: Release 8.0.5.0.0 - Production on Tue Jan 18 18:17:32 2000



Control File: T:/CTL/subacnpf1_ctl.txt

Data File: T:/CTL/subacnpf1_ctl.txt

Bad File: T:/CTL/subacnpf1_ctl.bad

Discard File: none specified

(Allow all discards)

Number to load: 10

Number to skip: 1

Errors allowed: 50

Bind array: 64 rows, maximum of 65536 bytes

Continuation: none specified

Path used: Conventional

Table SUBACNPF1, loaded from every logical record.

Insert option in effect for this table: INSERT

| Column Name | Position | Len | Term | Encl | Datatype |
|---|---|---|---|---|---|
| BNKKD | FIRST | * | , | O(") | CHARACTER |
| SBKOD | NEXT | * | , | O(") | CHARACTER |
| ACENT | NEXT | * | , | O(") | CHARACTER |

Table SUBACNPF1:

10 Rows successfully loaded.

0 Rows not loaded due to data errors.

0 Rows not loaded because all WHEN clauses were failed.

0 Rows not loaded because all fields were null.

Space allocated for bind array: 65016 bytes(84 rows)

Space allocated for memory besides bind array: 0 bytes

Total logical records skipped: 0

Total logical records read: 10

Total logical records rejected: 0

Total logical records discarded: 0

Run began on Tue Jan 18 18:17:32 2000

Run ended on Tue Jan 18 18:17:32 2000

Elapsed time was: 00:00:00.16

CPU time was: 00:00:00.05

## Ek 5 Export işlemi sonucu oluşan log dosyası

Connected to: Oracle8i Enterprise Edition Release 8.1.5.0.0 - Production

With the Partitioning and Java options

PL/SQL Release 8.1.5.0.0 - Production

Export done in WE8ISO8859P9 character set and WE8ISO8859P9 NCHAR character set

About to export specified users ...

. exporting foreign function library names for user GNL

. exporting object type definitions for user GNL

About to export GNL's objects ...

. exporting database links

. exporting sequence numbers

. exporting cluster definitions

. about to export GNL's tables via Conventional Path ...

| | | |
|---|---|---|
| . exporting table | AA | 2 rows exported |
| . exporting table | ACENTE_DEGISIM_TABLE | 0 rows exported |
| . exporting table | YKB_EFT_TABLE | 0 rows exported |
| . exporting table | YKB_TABLE | 10 rows exported |

.....

. exporting synonyms

. exporting views

. exporting stored procedures

. exporting referential integrity constraints

. exporting triggers

. exporting posttables actions

. exporting snapshots

. exporting snapshot logs

. exporting job queues

. exporting refresh groups and children

Export terminated successfully with warnings.

**ÖZGEÇMİŞ**

| | | |
|---|---|---|
| Doğum tarihi | 02.08.1976 | |
| Doğum yeri | İstanbul | |
| Lise | 1990-1993 | İstanbul Yeşilköy 50. Yıl Lisesi |
| Lisans | 1993-1997 | Yıldız Teknik Üniversitesi Kimya-Metalurji Mühendislik Fak.<br>Matematik Mühendisliği Bölümü |
| Yüksek Lisans | 1997-2001 | Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü<br>Matematik Müh. Anabilim Dalı, Matematik Mühendisliği Programı |

**Çalıştığı kurum**

| | | |
|---|---|---|
| | 1998- | Başak Hayat Sigorta A.Ş. |