1.Ünite - Kurulum ve PL/SQL'e Giriş

---

1. Giriş

PL/SQL (Procedural Language/SQL), Oracle tarafından geliştirilen **prosedürel veritabanı programlama dili**dir.
Amaç, SQL sorgularına **koşul, döngü, hata yönetimi** gibi programlama yapıları kazandırmaktır.
Dil yapısı **Ada** ve **Pascal** dillerine benzer.

PL/SQL dili Oracle Database 6.0 (1991) sürümünde ortaya çıkmış, zamanla **prosedür, fonksiyon, trigger, paket** gibi yapılar eklenmiştir.
Oracle dışında da benzer sürümler geliştirilmiştir:

- IBM DB2 → PL/SQL desteği (v9.7+)
- PostgreSQL → PL/pgSQL
- MariaDB → PL/SQL alt kümesi (v10.3+)

---

2. Gerekli Yazılımlar ve Kurulum

PL/SQL uygulamaları için gerekli bileşenler:

Yazılım	Amaç	Not
Oracle VirtualBox	Sanal makine çalıştırma	Windows üzerinde Oracle Linux imajı için gerekli
Oracle Database 23c Free (Developer Release)	Oracle veritabanı	Linux tabanlı sanal imaj (.ova) kullanılır
Oracle SQL Developer	SQL ve PL/SQL editörü	Oracle hesabı ile indirilir
VirtualBox Extension Pack	Ek sürücü ve ağ desteği	VirtualBox ile birlikte kurulmalıdır

Sanal Makine Özellikleri

- **OS:** Oracle Linux 8.7
- **DB:** Oracle Database 23.2 Free
- **Araçlar:** Oracle REST Data Services, APEX, SQLcl

Minimum sistem gereksinimleri:
4 GB RAM, 20 GB disk, 2 CPU çekirdeği
(önerilen: 8 GB RAM ve üzeri)

Oracle SQL Developer Kurulumu

- Zip dosyası indirildikten sonra açılır.
- `sqldeveloper.exe` çalıştırılarak başlatılır.
- Menü görüntü problemi varsa **Tools → Preferences → Oracle → Windows** seçilmelidir.

---

3. Oracle Bağlantı Kurulumu

Tarayıcı Üzerinden Bağlantı

1. Tarayıcıdan `http://localhost:8080/ords/sql-developer` adresi açılır.
2. Kullanıcı adı: `HR`
3. Şifre: `oracle`
4. Örnek sorgu:

```
sql SELECT * FROM COUNTRIES WHERE REGION_ID = 1;
```

Oracle SQL Developer ile Bağlantı

1. **Connections → + (New Connection)**
2. Bilgiler:
    - Name: Oracle Sanal Makine
    - Username: HR
    - Password: oracle
    - Hostname: localhost
    - Port: 1521
    - Service name: freepdb1
3. **Test** → “Success” mesajı alınırsa bağlantı tamamdır.
4. Aynı yöntemle diğer şemalar (OE, PM, IX, BI, SH) da eklenebilir.

---

4. PL/SQL’e Giriş

PL/SQL, SQL’in yetersiz kaldığı yerlerde **kontrol yapıları, hata yönetimi, modülerlik** sağlar.
Her PL/SQL komutu **noktalı virgül (;)** ile biter.
Tip güvenli, taşınabilir ve modüler bir dildir.

PL/SQL’in Sağladıkları

- **Prosedürler** ve **fonksiyonlar**
- **Koşullar** (IF, CASE)
- **Döngüler** (LOOP, FOR, WHILE)
- **İstisna yönetimi (EXCEPTION)**
- **Paketler** (package / package body)
- **Tip güvenliği** ve derleme zamanı kontrolü

Avantajları

- Sunucu tarafında çalıştığı için hızlıdır.
- Modüler yapı kod bakımını kolaylaştırır.
- Farklı platformlarda taşınabilir.
- Java, C#, PHP gibi dillerden çağrılabilir.

---

5. PL/SQL Kod Mimarisi

PL/SQL kodları **iki ana yapıdadır**:

Tür	Açıklama	Özellik
Anonim Blok	Veri tabanına kaydedilmez, isimsizdir.	Her çalıştırmada yeniden derlenir.
İsimli Blok	Veri tabanına kaydedilir (örneğin prosedür, fonksiyon, paket).	Derlenmiş hali saklanır, hızlı çalışır.

5.1. Anonim Blok Yapısı

```
DECLARE
  -- değişken tanımlamaları
BEGIN
  -- çalıştırılabilir ifadeler
```

```
EXCEPTION
  -- hata yönetimi
END;
```

Bölümler:

1. **DECLARE** → Değişkenlerin tanımlandığı bölüm (isteğe bağlı)
2. **BEGIN … END** → Çalıştırılabilir komutlar (zorunlu)
3. **EXCEPTION** → Hataların yakalandığı bölüm (isteğe bağlı)

Örnek:

```
DECLARE
  v_name VARCHAR2(20);
BEGIN
  v_name := 'Ahmet';
  DBMS_OUTPUT.PUT_LINE('Merhaba ' || v_name);
END;
```

- İç içe bloklar (nested blocks) oluşturulabilir.
- Bloklara etiket verilebilir (<<etiket>>).

5.2. İsimli Blok Yapısı

Yapı Türü	Açıklama	Örnek
Prosedür (Procedure)	Geri değer döndürmez	CREATE PROCEDURE selamla IS ...
Fonksiyon (Function)	RETURN ile değer döndürür	CREATE FUNCTION topla RETURN NUMBER IS ...
Paket / **Paket Gövdesi**	Birden fazla altprogramı toplar	CREATE PACKAGE personel_pkg AS ...
Trigger / **Type** / **Library**	Veritabanı olaylarını veya veri tiplerini yönetir	CREATE TRIGGER ...

Avantajları:

- Performans (önceden derlenmiş halde saklanır)
- Modülerlik ve kodun tekrar kullanılabilirliği
- Taşınabilirlik ve kolay bakım

---

6. PL/SQL Girdi/Çıktı Komutları

Konsola Yazdırma

```
SET SERVEROUTPUT ON;
BEGIN
  DBMS_OUTPUT.PUT_LINE('Merhaba PL/SQL!');
END;
```

- DBMS_OUTPUT.PUT_LINE → Satır sonu ekleyerek yazdırır.
- DBMS_OUTPUT.PUT → Satır sonu eklemeden yazar.

  Bu komutlar genelde test ve hata ayıklama aşamasında kullanılır.

Kullanıcıdan Girdi Alma

```
ACCEPT sayi NUMBER PROMPT 'Bir sayı giriniz: '
DECLARE
  v_num NUMBER := &sayi;
BEGIN
  DBMS_OUTPUT.PUT_LINE('Girdiğiniz sayı: ' || v_num);
END;
```

- & operatörü kullanıcıdan girdi alır.

Sonuç

- Oracle Database 23c Free sanal makine ile çalıştırılır.
- PL/SQL, SQL diline prosedürel özellikler kazandırır.
- Kodlar **anonim** veya **isimli blok** biçiminde yazılır.
- Üç ana bölüm: **DECLARE**, **BEGIN**, **EXCEPTION**.
- DBMS_OUTPUT.PUT_LINE ile çıktı alınır, & ile girdi istenir.
- İsimli bloklar derlenmiş biçimde saklandığı için performans sağlar.

2.Ünite - PL/SQL Temelleri

PL/SQL'de Kullanılan Karakterler

PL/SQL, veritabanı karakter setinde yer alan harfler (A-Z, a-z), rakamlar (0-9), noktalama işaretleri ve beyaz boşluklardan oluşur. Bu karakterler, komut ve ifadeleri oluşturmak için kullanılır.

PL/SQL Sözcük Türleri

PL/SQL dilinde sözcük türleri; ayraçlar, tanıtıcılar, literaller, değişkenler ve sabitler, ayrılmış ve anahtar kelimeler ile yorum satırlarıdır.

Ayraçlar (Delimiters)

Kod içinde özel işlevi olan semboller veya karakter gruplarıdır. Örnekler:

- := atama,
- || birleştirme,
- /* ... */ çok satırlı yorum,
- -- tek satırlı yorum,
- ; komut sonlandırıcı,
- ** üs alma operatörü.

Tanıtıcılar (Identifiers)

Programcı tarafından değişken, sabit, fonksiyon, paket gibi nesnelere verilen isimlerdir. Kurallar:

- Harf ile başlar, boşluk ve yasak karakter içermez.
- $, _, # kullanılabilir.
- Türkçe karakter kullanılmaz.
- PL/SQL'e özgü ayrılmış kelimeler tanıtıcı olarak kullanılamaz.
- Maksimum uzunluk 128 bayttır. Tanıtıcılar büyük/küçük harfe duyarsızdır; ancak çift tırnak içinde kullanıldığında duyarlı hale gelir.

Görünürlük (Scope): Bir tanıtıcı yalnızca tanımlandığı blokta geçerlidir. Dış blokta tanımlanan değişken iç blokta erişilebilir; ancak iç blokta tanımlanan değişken dış bloktan erişilemez.

Literaller

Sabit değerleri ifade eder. Türleri:

- **Sayı:** 1453, -15072016, 36E5
- **Karakter:** 'PL/SQL', 'A'
- **Tarih:** '01-Dec-1980'
- **Zaman Aralığı:** '123-2'
- **Boolean:** TRUE, FALSE, NULL

Kaçış karakterleri: Tek tırnak içindeki tek tırnaklar'' veya q'! ... !' biçimiyle kaçırılır.

Değişkenler ve Sabitler

- **Değişkenler:** Bellekte veri saklamak için kullanılır. Veri tipi belirtilmelidir.

plsql DECLARE sayi NUMBER := 10; * **Sabitler:** Değeri değiştirilemez. CONSTANT ile tanımlanır.

plsql pi CONSTANT NUMBER := 3.14;

Ayrılmış ve Anahtar Kelimeler

- **Ayrılmış Kelimeler:** PL/SQL'in kendi sözdizimi için kullanılır; tanıtıcı olarak kullanılamaz. Örn: BEGIN, END, SELECT, IF, NULL
- **Anahtar Kelimeler:** Tanıtıcı olarak kullanılabilir, fakat önerilmez. Örn: ADD, BODY, DATE, DELETE, UPDATE

Yorum Satırları

- Tek satır: -- açıklama
- Çok satır: /* açıklama */

PL/SQL Operatörleri

Atama Operatörü

:= operatörü, sağdaki değeri soldaki değişkene atar.

Birleştirme Operatörü

|| operatörü karakterleri veya dizgeleri birleştirir. Örn:'Merhaba' || ' Dünya'

Aritmetik Operatörler

+, -, *, /, ** Üs alma: a ** b veya POWER(a,b) Mod alma: MOD(a,b)

Karşılaştırma Operatörleri

=, !=, <>, <, >, <=, >= Eşitlik, büyüklük, küçüklük ilişkilerini kontrol eder.

Mantıksal Operatörler

AND, OR, NOT Koşul ifadelerinde mantıksal bağlantılar kurmak için kullanılır.

Sonuç

PL/SQL dilinin temellerini oluşturan karakter seti, sözcük türleri, tanıtıcı kuralları, literaller, değişken-sabit tanımlamaları, yorum satırları ve operatörleri öğretir. Bu bilgiler, sonraki ünitelerdeki PL/SQL blok yapısı, akış kontrolü ve hata yönetimi konularını anlamak için ön koşuldur.

3.Ünite - PL/SQL Veri Tipleri

---

1. Genel Bakış

PL/SQL'de veri tipleri, verilerin **nasıl saklanacağını ve işleneceğini** belirler.
İki ana gruba ayrılır:

- **Skaler (Scalar)**: Tek bir değer tutar (ör. sayı, karakter, tarih)
- **Bütünleşik (Composite)**: Birden fazla veri içerir (ör. tablo, kayıt türleri)

PL/SQL, hem **Oracle SQL veri tiplerini** hem de kendine özel veri tiplerini destekler.
Örneğin: PLS_INTEGER, BINARY_INTEGER, REF CURSOR sadece PL/SQL'e özeldir.

---

2. Karakter Veri Tipleri

Alfanümerik (harf + rakam) verileri saklamak için kullanılır.

Veri Tipi	Açıklama	PL/SQL Maks. Boyut	SQL Maks. Boyut

Veri Tipi	Açıklama	PL/SQL Maks. Boyut	SQL Maks. Boyut
CHAR	Sabit uzunlukta karakter dizisi	32767 bayt	2000 bayt
NCHAR	Sabit uzunlukta *ulusal karakter seti*	32767 bayt	2000 bayt
VARCHAR2	Değişken uzunlukta karakter dizisi	32767 bayt	4000–32767 bayt
NVARCHAR2	Değişken uzunlukta *ulusal karakter seti*	32767 bayt	4000–32767 bayt

CHAR

- Sabit uzunlukludur, kalan kısımlar **boşluk karakteriyle** doldurulur.

  sql ch CHAR(5) := 'A'; -- Bellekte "A " olarak saklanır.

VARCHAR2

- Değişken uzunluktadır, **boşluk eklenmez**.

  sql v VARCHAR2(20) := 'Ahmet';

NCHAR / NVARCHAR2

- Uluslararası karakterleri (ör. Türkçe, Çince) doğru saklamak için kullanılır.

---

3. Nümerik Veri Tipleri

Sayıları (tam veya ondalıklı) saklar.

3.1. NUMBER

- Tüm platformlarda **taşınabilir** ve en yaygın kullanılan tiptir.
- 1–22 bayt arası yer kaplar.

  sql salary NUMBER(8,2); -- 6 tam + 2 ondalık basamak

Alt Tip	Tür	Açıklama
INTEGER, INT, SMALLINT	Tam sayı	38 basamağa kadar
DEC, DECIMAL, NUMERIC	Sabit noktalı	38 basamağa kadar ondalıklı
FLOAT, DOUBLE PRECISION, REAL	Kayan noktalı	63–126 bit hassasiyet

3.2. FLOAT

- Kayan noktalı sayıları depolar, **yüksek hassasiyet** sağlar.

3.3. BINARY_FLOAT / BINARY_DOUBLE

- Donanım tabanlı, **hızlı matematiksel işlem** için kullanılır.
- Yaklaşık sonuçlar döndürebilir (örneğin 0.1 + 0.2 = 0.30000004).

3.4. PLS_INTEGER / BINARY_INTEGER

- -2.147.483.648 ↔ 2.147.483.647 aralığında değer alır.
- **NUMBER'dan daha hızlıdır** (işlemci üzerinden hesaplanır).
- 4 bayt yer kaplar.

Alt Tip	Açıklama
NATURAL	0–2.147.483.647, NULL olabilir
NATURALN	0–2.147.483.647, NULL olamaz

Alt Tip	Açıklama
POSITIVE	1–2.147.483.647, NULL olabilir
POSITIVEN	1–2.147.483.647, NULL olamaz
SIGNTYPE	-1, 0, 1, NULL değerlerini alır
SIMPLE_INTEGER	Aynı aralıkta, NULL alamaz, en hızlı tip

---

4. Tarih ve Saat Veri Tipleri

Veri Tipi	Açıklama
DATE	Tarih + saat (saniyeye kadar)
TIMESTAMP	Daha hassas (milisaniye düzeyi)
TIMESTAMP WITH TIME ZONE	Saat dilimi içerir
TIMESTAMP WITH LOCAL TIME ZONE	Kullanıcı saat dilimine göre kaydeder
INTERVAL	Zaman aralığı saklar

Önemli Fonksiyonlar

Fonksiyon	Açıklama
SYSDATE	Sunucunun tarih/saat bilgisi
CURRENT_DATE	Oturumun tarih/saat bilgisi
CURRENT_TIMESTAMP	Oturumun zaman damgası (saat dilimi dahil)
SYSTIMESTAMP	Sunucunun zaman damgası
LOCALTIMESTAMP	Yerel zaman damgası

---

5. INTERVAL Veri Tipleri

Zaman aralıklarını tutar.

5.1. INTERVAL YEAR TO MONTH

- Yıl ve ay cinsinden zaman farkı saklar.

  sql INTERVAL '2-6' YEAR TO MONTH -- 2 yıl 6 ay

5.2. INTERVAL DAY TO SECOND

- Gün, saat, dakika ve saniye cinsinden zaman farkı saklar.

  sql INTERVAL '5 12:30:45' DAY TO SECOND -- 5 gün 12 saat 30 dk 45 sn

---

6. BOOLEAN Veri Tipi

- **Mantıksal değerler** tutar: TRUE, FALSE, NULL.
- Oracle tablolarında kullanılmaz, sadece **PL/SQL değişkenleri** için geçerlidir.

  sql DECLARE kontrol BOOLEAN := TRUE; BEGIN IF kontrol THEN DBMS_OUTPUT.PUT_LINE('Evet'); END IF; END;

---

7. LOB (Large Object) Veri Tipleri

Büyük boyutlu veriler (metin, dosya, resim, video vb.) için kullanılır.
32 KB sınırını aşan verilerde tercih edilir.

Veri Tipi	Açıklama
CLOB	Karakter verisi (veritabanı karakter seti)

Veri Tipi	Açıklama
NCLOB	Ulusal karakter seti
BLOB	Binary (resim, ses, video) veriler
BFILE	Dosyanın içeriğini değil, **dosya yolunu** saklar

---

8. %TYPE Özniteliği

Bir değişkenin veri tipini, bir tablo kolonuna **otomatik olarak bağlar**.
Kolon tipi değişirse kodda değişiklik gerekmez.

```
DECLARE
  v_region_id regions.region_id%TYPE;
  v_region_name regions.region_name%TYPE;
BEGIN
  SELECT region_id, region_name
  INTO v_region_id, v_region_name
  FROM regions
  WHERE region_id = 4;
END;
```

---

9. Veri Tipi Dönüşümleri

9.1. Örtük (Implicit)

- PL/SQL otomatik dönüştürür.

  sql num NUMBER := '123'; -- string → number txt VARCHAR2(10) := 456; -- number → string

9.2. Aşikâr (Explicit)

- Kullanıcı dönüşüm fonksiyonu belirtir.

Fonksiyon	Dönüşüm
TO_CHAR()	Sayı/tarih → karakter
TO_NCHAR()	Sayı/tarih → ulusal karakter
TO_NUMBER()	Karakter → sayı
TO_DATE()	Karakter → tarih
TO_TIMESTAMP()	Karakter → timestamp
TO_TIMESTAMP_TZ()	Karakter → timestamp (saat dilimiyle)

Sonuç

- PL/SQL veri tipleri: **karakter, sayısal, tarih/zaman, mantıksal, büyük veri nesneleri**
- %TYPE özniteliği, dinamik ve bakımı kolay kod sağlar.
- Veri tipi dönüşümleri **örtük** veya **aşikâr** şekilde yapılır.
- BOOLEAN, PLS_INTEGER, BLOB, INTERVAL gibi türler **PL/SQL'e özgüdür**.

4.Ünite - Koşul ve Döngü Yapıları

---

1. Giriş

PL/SQL'de **koşullar ve döngüler**, program akışını yönlendiren temel yapılardır.

- **Koşul yapıları** → Programın belirli durumlara göre farklı işlemler yapmasını sağlar.
- **Döngüler** → Belirli işlemleri tekrar tekrar çalıştırır.

- **Sıralı kontrol deyimleri** (GOTO, NULL) → Akış kontrolünü özel durumlarda düzenler.

---

2. Koşul Yapıları

Koşul (kontrol) yapıları, **BOOLEAN** (TRUE / FALSE / NULL) değerine göre karar verir.
Sadece TRUE dönen koşulların içindeki kodlar çalıştırılır.

PL/SQL'de iki temel koşul yapısı vardır:

1. IF yapısı
2. CASE yapısı

2.1. IF Koşul Yapısı

IF-THEN

- En basit yapı.
- Koşul sağlanırsa blok çalışır; aksi halde atlanır.

```
IF sayi < 0 THEN
  DBMS_OUTPUT.PUT_LINE('Negatif sayı girdiniz.');
END IF;
```

IF-THEN-ELSE

- Koşul sağlanmazsa ELSE bloğu çalışır.

```
IF sayi > 0 THEN
  DBMS_OUTPUT.PUT_LINE('Pozitif');
ELSE
  DBMS_OUTPUT.PUT_LINE('Negatif veya Sıfır');
END IF;
```

IF-THEN-ELSIF

- Birden fazla koşul kontrolü için kullanılır.

```
IF sayi > 0 THEN
  DBMS_OUTPUT.PUT_LINE('Pozitif');
ELSIF sayi < 0 THEN
  DBMS_OUTPUT.PUT_LINE('Negatif');
ELSE
  DBMS_OUTPUT.PUT_LINE('Sıfır');
END IF;
```

> **Not:** NULL değerli değişkenlerle yapılan karşılaştırmalar her zaman**NULL** döndürür.
> Bu durumda IF bloğu çalışmaz, ELSE bloğu yürütülür.

2.2. CASE Koşul Yapısı

Basit CASE

Bir değişkenin belirli değerlerle eşleşmesini kontrol eder.

```
CASE sayi
  WHEN 0 THEN DBMS_OUTPUT.PUT_LINE('Sıfır');
  WHEN 1 THEN DBMS_OUTPUT.PUT_LINE('Bir');
  ELSE DBMS_OUTPUT.PUT_LINE('Bilinmiyor');
END CASE;
```

Aramalı (Searched) CASE

Koşullar açıkça belirtilir.

```
CASE
  WHEN sayi MOD 2 = 0 THEN DBMS_OUTPUT.PUT_LINE('Çift');
  ELSE DBMS_OUTPUT.PUT_LINE('Tek');
```

```
END CASE;
```

> CASE yapısı, çoklu IF bloklarına göre daha okunaklıdır.

---

3. Döngü Yapıları

Döngüler, belirli koşullar sağlandığı sürece kodun tekrar çalışmasını sağlar.
PL/SQL'de 3 temel döngü vardır:

1. **LOOP** (temel döngü)
2. **FOR döngüsü**
3. **WHILE döngüsü**

Ayrıca:

- **İç içe döngüler (nested loops)**
- **Döngü etiketleri**
- **EXIT**, **EXIT WHEN**, **CONTINUE**, **CONTINUE WHEN** deyimleri ile kontrol sağlanır.

3.1. LOOP Döngüsü

Sınırsız döngüdür, koşul verilmez.
Sonlandırmak için EXIT veya EXIT WHEN kullanılır.

```
DECLARE
  i NUMBER := 1;
BEGIN
  LOOP
    DBMS_OUTPUT.PUT_LINE(i);
    i := i + 1;
    EXIT WHEN i > 10;
  END LOOP;
END;
```

> Eğer EXIT eklenmezse **sonsuz döngü** oluşur.

3.2. FOR Döngüsü

Başlangıç ve bitiş değerleri bellidir.

```
FOR i IN 1..5 LOOP
  DBMS_OUTPUT.PUT_LINE(i);
END LOOP;
```

Geriye doğru sayım için REVERSE kullanılır:

```
FOR i IN REVERSE 10..1 LOOP
  DBMS_OUTPUT.PUT_LINE(i);
END LOOP;
```

> Döngü indisi (i) sadece döngü bloğu içinde geçerlidir.
> Döngü bitince erişilemez.

3.3. WHILE Döngüsü

Koşul doğru olduğu sürece döngü çalışır.

```
DECLARE
  i NUMBER := 1;
BEGIN
  WHILE i <= 5 LOOP
    DBMS_OUTPUT.PUT_LINE(i);
    i := i + 1;
  END LOOP;
END;
```

Koşul başlangıçta FALSE ise döngü hiç çalışmaz.

---

4. İç İçe Döngüler (Nested Loops) ve Etiketler

Bir döngü içinde başka bir döngü çalıştırılabilir.
Her döngünün sayaç değişkeni farklı olmalıdır.

```
FOR i IN 1..3 LOOP
  FOR j IN 1..3 LOOP
    DBMS_OUTPUT.PUT_LINE('i='||i||', j='||j);
  END LOOP;
END LOOP;
```

Etiket Kullanımı

Okunabilirliği artırır, karmaşık döngülerde özellikle faydalıdır.

```
<<dis_dongu>>
FOR i IN 1..5 LOOP
  <<ic_dongu>>
  FOR j IN 1..5 LOOP
    EXIT dis_dongu WHEN j = 3;
  END LOOP ic_dongu;
END LOOP dis_dongu;
```

Dış döngü, içteki döngü tarafından **etiketle sonlandırılabilir**.
Ancak tersi (iç döngünün dıştan kapatılması) mümkün değildir.

---

5. Sıralı Kontrol Deyimleri

PL/SQL, akışı özel durumlarda yönlendirmek için iki deyim sunar:

5.1. GOTO

Program akışını belirtilen etikete yönlendirir.
Ancak **tavsiye edilmez** (kodun okunurluğunu düşürür).

```
BEGIN
  GOTO atla;
  DBMS_OUTPUT.PUT_LINE('Bu satır çalışmaz.');
  <<atla>>
  DBMS_OUTPUT.PUT_LINE('GOTO etiketi atlandı.');
END;
```

5.2. NULL

Hiçbir işlem yapmaz, genellikle **boş koşul** için kullanılır.

```
IF sayi = 0 THEN
  NULL; -- hiçbir şey yapılmaz
ELSE
  DBMS_OUTPUT.PUT_LINE('Sıfır değil');
END IF;
```

Sonuç

- **Koşullar**: IF, ELSIF, CASE
- **Döngüler**: LOOP, FOR, WHILE
- **Kontrol deyimleri**: EXIT, CONTINUE, GOTO, NULL
- İç içe döngülerde **etiketler** okunabilirliği artırır.
- EXIT WHEN koşulu sağlanmazsa **sonsuz döngü** oluşur.

5.Ünite - Kursörler

Veritabanı sorgularından dönen sonuçların satır satır işlenmesini sağlayan kursör (imleç) yapısını ve detaylarını ele almaktadır.

1. Kursör (İmleç) Temel Kavramları

- **Tanım:** Sorgu sonuçlarını bir döngü aracılığıyla tek tek işlemeye yarayan PL/SQL bileşenidir.
- **Aktif Set (Active Set):** Oracle, sorgu sonuçlarını tutmak için özel bir bellek alanı ayırır; bu alana "aktif set" denir. Kursör bu setin en başına konumlanır ve satır satır ilerler.
- **Avantajları:**
  - Çok satırlı sorgu sonuçlarının yönetilmesini sağlar.
  - Veri istemciye çekilmeden sunucu belleğinde işlendiği için performans sağlar.
  - Veri bütünlüğünü korumak için satır kilitleme (locking) yapılabilir.

2. Kursör Türleri

PL/SQL'de iki ana kursör türü bulunur:

A. Örtük (Implicit) Kursörler

- **Otomatik Oluşum:** `INSERT`, `UPDATE`, `DELETE` gibi DML komutları veya tek satır döndüren `SELECT` sorguları çalıştığında sunucu tarafından otomatik oluşturulur ve kapatılır.
- **Yönetim:** Programcı müdahale edemez, sadece niteliklerini okuyabilir.
- **Nitelikler:**
  - `SQL%ISOPEN`: Örtük kursörlerde her zaman `FALSE` döner çünkü işlem bitince hemen kapanır.
  - `SQL%FOUND` / `SQL%NOTFOUND`: İşlem sonucunda satır dönüp dönmediğini belirtir.
  - `SQL%ROWCOUNT`: İşlemden etkilenen satır sayısını verir.

B. Aşikâr (Explicit) Kursörler

- **Manuel Kontrol:** Birden fazla satır döndüren sorgular için programcı tarafından `DECLARE` bloğunda tanımlanır.
- **Yaşam Döngüsü:** 4 aşamadan oluşur:
  1. **Tanımlama (Declare):** `CURSOR kursör_adı IS SELECT...` şeklinde yapılır.
  2. **Açma (Open):** `OPEN kursör_adı;` komutuyla aktif set oluşturulur.
  3. **Veri Çekme (Fetch):** `FETCH kursör_adı INTO değişkenler;` ile veriler satır satır alınır.
  4. **Kapatma (Close):** `CLOSE kursör_adı;` ile bellek serbest bırakılır.
- **Performans:** Sorgu sadece ilk açılışta çözümlenir (parse edilir), kapatılıp tekrar açıldığında tekrar çözümleme yapılmaz.
- **Parametre Kullanımı:** Kursörler parametre alarak daha esnek hale getirilebilir.

3. Kullanım Yöntemleri ve İpuçları

- **FOR Döngüsü:** Aşikâr kursörler `FOR` döngüsü ile kullanıldığında; açma, veri çekme ve kapatma işlemleri otomatik yapılır, kod kısalır.
- **SELECT FOR UPDATE:** Veriler işlenirken başka oturumların bu verileri değiştirmesini engellemek (kilitlemek) için kullanılır.
  - `NOWAIT`: Kilitli satır varsa beklemeden hata fırlatır.
  - `WAIT n`: Kilidin kalkması için n saniye bekler.
  - `SKIP LOCKED`: Kilitli satırları atlar.

4. Kursör Değişkenleri (Ref Cursor)

- **Referans Mantığı:** Doğrudan sonucu değil, sonucun bulunduğu bellek adresini referans alır.
- **Esneklik:** Bir kez tanımlandıktan sonra farklı sorgular için tekrar tekrar kullanılabilir ve alt programlara parametre olarak gönderilebilir.
- **Türleri:**
  - **Zayıf (Weak):** Dönüş tipi belli değildir (SYS_REFCURSOR veya REF CURSOR). Her türlü sorguyla ilişkilendirilebilir.
  - **Kuvvetli (Strong):** RETURN ile dönüş tipi belirtilir, sadece uyumlu sorgularla çalışır.

6.Ünite - Kayıt ve Koleksiyon Veri Tipi

Tek bir değer tutan değişkenlerin ötesine geçerek, birden fazla veriyi mantıksal bir yapı altında saklamayı sağlayan **Bütünleşik Veri Yapılarını** (Record ve Collection) ele almaktadır.

1. Bütünleşik Veri Yapıları

Tek seferde tek bir veri tutan skaler tiplerin (VARCHAR2, NUMBER vb.) aksine, bütünleşik veri tipleri veriyi bir grup halinde yönetmeyi sağlar. Bu yapılar kodun okunabilirliğini artırır ve bakım maliyetlerini düşürür. İki temel yapı vardır: **Record (Kayıt)** ve **Collection (Koleksiyon)**.

2. Kayıt Veri Tipi (Record)

Farklı veri tiplerindeki verilerin (örneğin bir öğrencinin adı, numarası ve doğum tarihi) tek bir mantıksal çatı altında tutulmasını sağlar. Bir veritabanı tablosundaki "satır" mantığına benzer.

- **Yapısı:** "Field" adı verilen bileşenlerden oluşur. Verilere nokta notasyonu (kayit_adi.alan_adi) ile erişilir.
- **Record Türleri:**
  1. **Programcı Tanımlı Record:** Alanların isimleri ve veri tipleri programcı tarafından TYPE ... IS RECORD bloğu içinde tek tek belirtilir.
  2. **Kursör Tabanlı Record:** Bir kursörün döndürdüğü sonuç kümesinin yapısını referans alır (%ROWTYPE kullanılır).
  3. **Tablo Tabanlı Record:** Doğrudan bir veritabanı tablosunun veya view'ın yapısını referans alır (tablo_adi%ROWTYPE). Tablodaki bir sütun değişirse, record yapısı da otomatik uyum sağlar.

3. Koleksiyon Veri Yapısı (Collection)

Dizi (array) mantığına benzer şekilde, **aynı türdeki** çok sayıda veriyi saklamak için kullanılır.

- **Temel Koleksiyon Türleri:**
  1. **Associative Array (İlişkisel Dizi):** Anahtar-değer (Key-Value) çifti mantığıyla çalışır. Boyutu dinamiktir, eleman sayısı baştan belirtilmez. İndisler sayısal veya metin (string) tabanlı olabilir. Sıralı olma zorunluluğu yoktur (seyrek yapı).
  2. **VARRAY (Değişken Boyutlu Dizi):** Boyutu (maksimum eleman sayısı) tanımlanırken belirtilir ve sabit sınırları vardır. Elemanlar sıralı ve bitişiktir (sık yapı).
  3. **Nested Table (İç İçe Tablo):** Boyutu sınırlandırılmamıştır, dinamik olarak büyüyüp küçülebilir. Tek boyutlu dizi gibi davranır ve indisler 1'den başlar.

4. Koleksiyon Metodları

Koleksiyonlar üzerinde işlem yapmak, gezinmek veya boyutlarını yönetmek için kullanılan fonksiyonlardır:

- **Gezinme:** FIRST (ilk eleman), LAST (son eleman), PRIOR (önceki), NEXT (sonraki).
- **Yönetim:** COUNT (mevcut eleman sayısı), LIMIT (maksimum kapasite - sadece VARRAY için), EXISTS (eleman var mı?).
- **Değişiklik:** EXTEND (yeni eleman için yer açar), TRIM (sondan eleman siler), DELETE (belirli veya tüm elemanları siler).

7.Ünite - Alt Program ve Paketler

Kod tekrarını önlemek, yönetimi kolaylaştırmak ve performansı artırmak amacıyla kullanılan **Alt Programlar** (Prosedür ve Fonksiyonlar) ile bunları gruplayan **Paket** yapılarını ele almaktadır.

1. Alt Programlar (Subprograms)

Sürekli tekrar eden kod bloklarının isimlendirilerek saklanmasıdır. Derlenmiş halde sunucuda tutuldukları için her çağrıldıklarında tekrar derlenmezler, bu da performans artışı sağlar.

- **Alt Program Türleri:**
    1. **Bağımsız (Standalone):** CREATE komutuyla doğrudan şema altında oluşturulan ve veritabanında saklanan türdür.
    2. **Paket Alt Programları:** Bir paket yapısı içinde tanımlananlardır.
    3. **İç İçe (Nested):** Bir PL/SQL bloğu içinde tanımlanan ve sadece o blokta geçerli olan türdür.
- **Prosedür (Procedure) ve Fonksiyon (Function) Ayrımı:**
    - **Prosedür:** Belirli bir işi (eylemi) yerine getirmek için kullanılır. Geriye bir değer döndürme zorunluluğu yoktur (ancak parametreler ile değer taşıyabilir). SQL sorguları (SELECT) içinde doğrudan kullanılamazlar.
    - **Fonksiyon:** Muhakkak geriye bir değer döndürmelidir (RETURN kullanımı zorunludur). Bir atama işleminde veya SQL sorgusu içinde kullanılabilirler.
- **Parametre Modları:**
    - **IN:** Varsayılan moddur. Parametre dışarıdan değer alır, alt program içinde kullanılır ancak değiştirilemez.
    - **OUT:** Parametre dışarıdan değer almaz, alt program içinde değer atanarak dışarıya sonuç gönderir.
    - **IN OUT:** Parametre hem dışarıdan başlangıç değeri alır hem de alt program içinde değiştirilerek güncel değeri dışarıya gönderir.

2. Paketler (Packages)

Birbiriyle ilişkili prosedür, fonksiyon, değişken, imleç (cursor) ve tipleri mantıksal bir çatı altında toplayan nesnelerdir. İki temel bileşenden oluşur:

1. **Paket Başlığı (Specification/Header):** Paketin "vitrini" veya arayüzüdür. Dışarıdan erişilebilecek değişkenler ve alt programların prototipleri (imzaları) burada tanımlanır. Kodun gövdesi (işleyişi) burada bulunmaz.
2. **Paket Gövdesi (Body):** Başlıkta tanımlanan alt programların asıl kodlarının (implementation) yazıldığı yerdir. Ayrıca sadece paket içinde geçerli olan (dışarıya kapalı/private) değişken ve alt programlar da burada tanımlanabilir.
3. **Avantajları:**
    - **Modülerlik:** İlişkili işlemler bir arada tutulur.
    - **Gizlilik:** Paket gövdesindeki özel tanımlar dışarıdan görülmez.
    - **Kalıcılık:** Paket değişkenleri, oturum boyunca değerlerini korurlar.
    - **Aşırı Yükleme (Overloading):** Aynı isimde fakat farklı parametre yapılarına (farklı veri tipi veya sayısı) sahip birden fazla alt programın tanımlanabilmesine olanak sağlar. Bu özellik sadece paketlerde kullanılabilir.

Örnekler

Örnek 1: Parametreli Prosedür (IN ve OUT Kullanımı)

Bu örnek, dışarıdan bir çalışan ID'si alır (IN) ve o çalışanın maaşını bulup dışarıya döndürür (OUT).

```
-- Prosedürün Tanımlanması
CREATE OR REPLACE PROCEDURE maas_bul (
   p_calisan_id IN NUMBER,      -- Dışarıdan veri alır
   p_maas OUT NUMBER            -- Dışarıya veri gönderir
) IS
BEGIN
   SELECT salary INTO p_maas
   FROM employees
   WHERE employee_id = p_calisan_id;
EXCEPTION
```

```
    WHEN NO_DATA_FOUND THEN
        p_maas := 0; -- Çalışan bulunamazsa 0 döndür
        DBMS_OUTPUT.PUT_LINE('Çalışan bulunamadı.');
END;
/

-- Prosedürün Çağrılması (Anonim Blok)
DECLARE
    v_gelen_maas NUMBER;
BEGIN
    -- 100 numaralı çalışanın maaşını bulup v_gelen_maas değişkenine atar
    maas_bul(100, v_gelen_maas);
    DBMS_OUTPUT.PUT_LINE('Çalışanın Maaşı: ' || v_gelen_maas);
END;
/
```

Örnek 2: Değer Döndüren Fonksiyon (Function)

Bu örnek, verilen bir yarıçap değerine göre dairenin alanını hesaplar ve sonucu`RETURN` ile döndürür.

```
-- Fonksiyonun Tanımlanması
CREATE OR REPLACE FUNCTION daire_alani_hesapla (
    p_yaricap NUMBER
) RETURN NUMBER IS
    v_pi CONSTANT NUMBER := 3.14159;
    v_alan NUMBER;
BEGIN
    v_alan := v_pi * POWER(p_yaricap, 2);
    RETURN v_alan; -- Hesaplanan değeri döndürür
END;
/

-- Fonksiyonun Kullanımı (SELECT içinde)
SELECT daire_alani_hesapla(5) AS Alan FROM DUAL;

-- Veya Anonim Blok İçinde
BEGIN
    DBMS_OUTPUT.PUT_LINE('Alan: ' || daire_alani_hesapla(10));
END;
/
```

Örnek 3: Paket (Package) ve Aşırı Yükleme (Overloading)

Bu örnek, ders notlarında vurgulanan "Paketlerde aynı isimde metod tanımlama" (Method Overloading) konusunu gösterir. `yazdir` isminde iki prosedür tanımlanır; biri sayıları, diğeri metinleri ekrana basar.

Adım 1: Paket Başlığı (Specification)

```
CREATE OR REPLACE PACKAGE yazdirma_araclari IS
    -- Aynı isimde iki farklı prosedür bildirimi
    PROCEDURE yazdir(p_veri NUMBER);
    PROCEDURE yazdir(p_veri VARCHAR2);
END yazdirma_araclari;
/
```

Adım 2: Paket Gövdesi (Body)

```
CREATE OR REPLACE PACKAGE BODY yazdirma_araclari IS

    -- Sayı alan versiyonun kodlanması
    PROCEDURE yazdir(p_veri NUMBER) IS
    BEGIN
        DBMS_OUTPUT.PUT_LINE('Sayısal Değer: ' || p_veri);
    END yazdir;

    -- Metin alan versiyonun kodlanması
    PROCEDURE yazdir(p_veri VARCHAR2) IS
    BEGIN
        DBMS_OUTPUT.PUT_LINE('Metinsel Değer: ' || p_veri);
    END yazdir;

END yazdirma_araclari;
/
```

Adım 3: Paketin Kullanımı

```
BEGIN
    -- PL/SQL hangi prosedürü çağıracağını veri tipinden anlar
    yazdirma_araclari.yazdir(1923);       -- Sayısal versiyon çalışır
```

```
    yazdirma_araclari.yazdir('Merhaba');  -- Metinsel versiyon çalışır
END;
/
```

8.Ünite - Trigger ve İstisna İşleme

Veritabanı olaylarına otomatik tepki veren **Trigger (Tetikleyici)** mekanizmasını ve çalışma zamanı hatalarını yönetmeyi sağlayan **İstisna İşleme (Exception Handling)** yapısını ele almaktadır.

BÖLÜM 1: Trigger (Tetikleyici)

Trigger, veritabanında gerçekleşen belirli bir olaya (INSERT, UPDATE, DELETE vb.) bağlı olarak, o olaydan önce (BEFORE) veya sonra (AFTER) sunucu tarafından **otomatik** olarak çalıştırılan özel prosedürlerdir.

1. Triggerların Özellikleri ve Kullanım Alanları

- **Otomatik Çalışma:** Prosedürler gibi kullanıcı tarafından manuel çağrılmazlar, tanımlandıkları olay gerçekleşince devreye girerler.
- **Yönetim:** DISABLE komutu ile geçici olarak durdurulabilir, ENABLE ile tekrar aktif edilebilirler.
- **Kullanım Alanları:**
    - Veri güvenliği ve denetimi (Auditing/Loglama).
    - Karmaşık veri bütünlüğü kontrolleri.
    - Otomatik değer atama (Örn: Doğum tarihinden yaş hesaplama).
    - Yetkisiz erişimi engelleme (Örn: Mesai saatleri dışında işlem yasağı).

2. Trigger Türleri ve Bileşenleri

- **DML Triggerları:** Tablo üzerinde veri değişikliği (INSERT, UPDATE, DELETE) yapıldığında çalışır.
    - **Deyim Seviyesi (Statement Level):** İşlemden kaç satır etkilenirse etkilensin sadece bir kez çalışır.
    - **Satır Seviyesi (Row Level):** İşlemden etkilenen her bir satır için ayrı ayrı çalışır (FOR EACH ROW ile tanımlanır).
- **Sistem Triggerları:** Veritabanı (STARTUP, SHUTDOWN) veya şema olaylarına (CREATE, DROP, ALTER) bağlı çalışır.
- **Compound (Bütünleşik) Trigger:** Tek bir trigger bloğu içinde hem işlem öncesi/sonrası hem de satır bazlı işlemleri birleştiren yapıdır.
- **Instead Of Trigger:** Genellikle View'lar üzerinde, gerçekleşen işlemin yerine başka bir işlem yapmak için kullanılır.

3. Koşullu Predikatlar

Trigger gövdesinde hangi işlemin yapıldığını anlamak için kullanılır:

- INSERTING: Ekleme işlemi mi?
- UPDATING: Güncelleme işlemi mi?
- DELETING: Silme işlemi mi?

BÖLÜM 2: İstisna İşleme (Exception Handling)

Programın çalışması sırasında meydana gelen hatalara (Runtime Errors) **istisna** denir. İstisnalar işlenmezse program aniden sonlanır ve veri tutarsızlığına yol açabilir.

1. İstisna İşleme Bloğu

Hataları yakalamak için BEGIN ... END bloğunun sonuna EXCEPTION kısmı eklenir.

- WHEN hata_adi THEN ... yapısı ile spesifik hatalar yakalanır.

- WHEN OTHERS THEN ... ile tanımlanmamış diğer tüm hatalar yakalanır.

2. İstisna Türleri

1. **Sistem İstisnaları (Oracle Tanımlı):**
    - **İsimli (Ön Tanımlı):** Sık karşılaşılan hatalardır. Örn: NO_DATA_FOUND (veri bulunamadı), ZERO_DIVIDE (sıfıra bölme hatası), TOO_MANY_ROWS (tek satır beklerken çok satır gelmesi).
    - **İsimsiz (Dahili):** Bir ismi olmayan, sadece hata kodu (ORA-xxxxx) olan hatalardır.PRAGMA EXCEPTION_INIT ile isim verilebilir veya SQLCODE ile yakalanabilir.
2. **Kullanıcı Tanımlı İstisnalar:** Programcının belirlediği iş kurallarına aykırı durumlarda (Örn: Stok miktarının negatife düşmesi) oluşturduğu istisnalardır.
    - Tanımlama: hata_adi EXCEPTION;
    - Fırlatma: RAISE hata_adi; veya RAISE_APPLICATION_ERROR kullanılır.

3. Hata Bilgisi Alma Fonksiyonları

- SQLCODE: Hatanın sayısal kodunu döndürür.
- SQLERRM: Hatanın açıklama mesajını döndürür.

BÖLÜM 3: Pratik Kod Örnekleri

Aşağıda trigger ve istisna işleme konularını pekiştiren, doğru sözdizimi ile hazırlanmış örnekler yer almaktadır.

Örnek 1: DML Trigger (Loglama İşlemi)

Bir çalışanın maaşı güncellendiğinde, eski ve yeni maaşı kaydeden bir trigger örneği.

```
-- Trigger Tanımlama
CREATE OR REPLACE TRIGGER trg_maas_kontrol
AFTER UPDATE OF salary ON employees -- Sadece salary kolonu güncellenirse çalış
FOR EACH ROW -- Her satır için çalış (Eski ve yeni değerlere erişmek için)
BEGIN
   -- Maaş değiştiyse ekrana bilgi yaz (veya log tablosuna ekle)
   DBMS_OUTPUT.PUT_LINE('Eski Maaş: ' || :OLD.salary ||
                        ' - Yeni Maaş: ' || :NEW.salary);
END;
/
```

Örnek 2: İstisna İşleme (Sıfıra Bölme ve Diğerleri)

Bir bölme işleminde olası hataları yakalayan blok.

```
DECLARE
   v_bolunen NUMBER := 100;
   v_bolen   NUMBER := 0;
   v_sonuc   NUMBER;
BEGIN
   -- Bölme işlemi deneniyor
   v_sonuc := v_bolunen / v_bolen;
   DBMS_OUTPUT.PUT_LINE('Sonuç: ' || v_sonuc);

EXCEPTION
   -- Sıfıra bölme hatası yakalanırsa
   WHEN ZERO_DIVIDE THEN
      DBMS_OUTPUT.PUT_LINE('Hata: Bir sayı sıfıra bölünemez!');

   -- Beklenmeyen başka bir hata olursa
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Beklenmeyen Hata Kodu: ' || SQLCODE);
      DBMS_OUTPUT.PUT_LINE('Hata Mesajı: ' || SQLERRM);
END;
/
```

Örnek 3: Kullanıcı Tanımlı İstisna

Bir çalışana negatif prim verilmesini engelleyen özel hata yönetimi.

```
DECLARE
   v_prim_miktari NUMBER := -500;
   ex_gecersiz_prim EXCEPTION; -- Kullanıcı tanımlı istisna
BEGIN
   IF v_prim_miktari < 0 THEN
      -- Hata fırlatılır
      RAISE ex_gecersiz_prim;
   END IF;

   DBMS_OUTPUT.PUT_LINE('Prim tanımlandı: ' || v_prim_miktari);

EXCEPTION
   WHEN ex_gecersiz_prim THEN
      -- Özel hata mesajı verilir
      RAISE_APPLICATION_ERROR(-20001, 'Hata: Prim miktarı negatif olamaz!');
END;
/
```