

# BM 440 Veri Tabanı Tasarımı ve Uygulamaları

Dr. Öğr. Üyesi Ekrem BAŞER

[ekrembaser@duzce.edu.tr](mailto:ekrembaser@duzce.edu.tr)

5. Hafta Ders Notu

# Alt Sorgular

- Bir SQL sorgusunun içinde **başka bir SQL sorgusunun** yer aldığı duruma denir.
- Bir **subquery (alt sorgu)**, ana sorgunun (main query) WHERE, FROM, SELECT veya HAVING kısmında yer alabilir.
- Oracle önce alt sorguyu çalıştırır, sonra ana sorgu o sonucu kullanır.

# Tek Satır Döndüren Alt Sorgu

- Alt sorgu **tek bir satır veya tek bir değer** döndürür (örneğin: AVG, MAX, MIN).
- **Kullanılan operatörler:** =, >, <, >=, <=, <>

```
SELECT first_name, salary
FROM employees
WHERE salary > (
    SELECT AVG(salary)
    FROM employees
);
```

# Çok Satır Döndüren Alt Sorgu

- Alt sorgu birden fazla satır döndürür.  
Bu durumda IN, ANY, ALL gibi operatörler kullanılır.

- IN** → Alt sorgudan dönen **herhangi bir değere eşitse** koşul sağlanır.
- ANY** → Alt sorgudan dönen **en az bir değere göre** koşulu sağlıyorsa doğru olur.
- ALL** → Alt sorgudan dönen **tüm değerlere göre** koşulu sağlıyorsa doğru olur.

```
SELECT first_name, department_id  
FROM employees  
WHERE department_id IN (  
    SELECT department_id  
    FROM departments  
    WHERE location_id = 1700  
)
```

# Çok Satır Döndüren Alt Sorgu

Operatör	Anlamı	Örnek	Açıklama
IN	Alt sorgudan dönen değerlerden herhangi biriyle eşleş	x IN (alt_sorgu)	listedeki herhangi birine eşit
> ANY	Alt sorgudan dönen en az bir değerden büyük	salary > ANY (alt_sorgu)	en küçük değerden büyük
< ANY	Alt sorgudan dönen en az bir değerden küçük	salary < ANY (alt_sorgu)	en büyük değerden küçük
> ALL	Alt sorgudan dönen tüm değerlerden büyük	salary > ALL (alt_sorgu)	en büyük değerden bile büyük
< ALL	Alt sorgudan dönen tüm değerlerden küçük	salary < ALL (alt_sorgu)	en küçük değerden bile küçük

# İlişkili Alt Sorgu

- Alt sorgu, ana sorgudaki her satır için **yeniden çalışır**.  
Yani alt sorgu, ana sorgudaki bir sütuna bağlımlıdır.

```
SELECT e1.first_name, e1.department_id, e1.salary  
FROM employees e1  
WHERE e1.salary > (  
    SELECT AVG(e2.salary)  
    FROM employees e2  
    WHERE e2.department_id = e1.department_id  
)
```

Her departman için ayrı ayrı ortalama maaş bulunur.  
Kendi departman ortalamasının üstünde maaş alan çalışanlar listelenir.

# İç İçe Alt Sorgular

- Bir alt sorgunun içinde başka bir alt sorgu olabilir.  
Oracle bunları **ıçten dışa doğru** çalıştırır.

```
SELECT first_name  
FROM employees  
WHERE department_id = (  
    SELECT department_id  
    FROM departments  
    WHERE location_id = (  
        SELECT location_id  
        FROM locations  
        WHERE city = 'London'  
    )  
);
```

- En içteki sorgu London şehrinin location\_id'sini bulur.
- Orta sorgu o konumun departmanlarını bulur.
- Ana sorgu o departmandaki çalışanları listeler.

# Inline View (FROM içinde Alt Sorgu)

- Alt sorgu, FROM ifadesi içinde **tablo gibi** kullanılır.  
Bu yönteme “**inline view**” denir.

```
SELECT department_id, AVG(salary) AS avg_salary    SELECT department_id, avg_salary
FROM employees                                     FROM (
GROUP BY department_id                           SELECT department_id, AVG(salary) AS avg_salary
HAVING AVG(salary) > 8000;                         FROM employees
                                                       GROUP BY department_id
                                                       )
                                                       WHERE avg_salary > 8000;
```

İçteki sorgu her departman için ortalama maaşı hesaplar.  
Dış sorgu ortalama maaşı 8000'den büyük olanları gösterir.

# Tek Değer Döndüren ve SELECT İçinde Kullanılan Sorgu

- Alt sorgu **tek bir değer döndürür** ve SELECT kısmında **sütun gibi** kullanılabilir.

```
SELECT first_name,  
       salary,  
       (SELECT AVG(salary) FROM employees) AS avg_salary  
FROM employees;
```

Her çalışan satırında ortalama maaş da görüntülenir.

Bu sorgu her satırda aynı değeri gösterir, çünkü alt sorgu sabittir.

# HAVING ile Subquery Kullanımı

- Gruplandırılmış verileri alt sorgu sonucu ile karşılaştırmak için kullanılır.

```
SELECT department_id, AVG(salary) AS avg_salary  
FROM employees  
GROUP BY department_id  
HAVING AVG(salary) > (  
    SELECT AVG(salary)  
    FROM employees  
) ;
```

Ortalama maaşı genel ortalamadan yüksek olan departmanları listeler.

# PL/SQL Oracle Veri Tabanı Programlama

- PL/SQL : “**Procedural Language / Structured Query Language**” ifadesinin kısaltmasıdır. “Yapısal Sorgulama Diline (SQL) eklenmiş prosedürel bir dil” anlamına gelir.
- PL/SQL, SQL komutları üzerine prosedür özellikleri katar. Dolayısıyla PL/SQL çalışabilmek için orta veya üst seviyede SQL komutlarına hakim olmamız gereklidir.
- Tıpkı SQL komutlarında olduğu gibi, PLSQL programları tüm arayüz programlarından (Java, C#, Python,...) çalıştırılabilir.

# PL/SQL Oracle Veri Tabanı Programlama

- Kodlama için özel bir editöre ihtiyaç yoktur. Not Defteri ile çalıştırılabilir.
- Öğrenilmesi kolay.
- Taşınabilir.
- Tekrar kullanılabilir.
- Aynı anda farklı programlardan çalıştırılabilir.
- Nesne tabanlı programlama mimarisini destekler.
- PL/SQL-> Oracle, T-SQL->MsSQL, PL/pgSQL->PostgreSQL

# PL/SQL Block Türleri

- PL/SQL'de **her şey bir blok (block) içinde** yazılır.  
Bir **block**, bir işi (örneğin hesaplama, veri ekleme, mesaj yazdırma) yapan **mantıksal bir kod parçasıdır**.

Her PL/SQLbloğu şu üç ana bölümden oluşur

```
DECLARE      -- (1) Değişkenlerin tanımlandığı yer (isteğe bağlı)
BEGIN        -- (2) Asıl işlemlerin yapıldığı yer (zorunlu)
EXCEPTION    -- (3) Hata durumlarının yakalandığı yer (isteğe bağlı)
END;
```

# PL/SQL Kod Yapısı

**DECLARE** (isteğe Bağlı)

Declaration of Variables, Constants,  
Cursors, etc

**BEGIN** (Zorunlu)

PL/SQL Executable Statements  
(En az bir tane çalışabilir kod yazılmalıdır)

**EXCEPTION** (isteğe bağlı)

PL/SQL Exception Handler Block

**END;** (Zorunlu)

# İsimsiz Blok (Anonymous Block)

- “İsimsiz blok” olarak geçer.
- Veritabanında kaydedilmez, sadece anlık çalışır.
- Genelde test, geçici sorgular veya küçük işlemler için kullanılır.
- Bu blok çalışır ama bittiğinde **bellekten silinir**.
- Doğrudan çalıştırılır.

```
BEGIN  
    DBMS_OUTPUT.PUT_LINE('Bu bir anonymous block.');//  
END;  
/
```

Not: Ekrana çıktı yazdırmak için bu komutu daha önceden çalıştırmanız gereklidir: SET SERVEROUTPUT ON;

# İsimli Blok (Named Block)

- Veritabanında kalıcı olarak saklanır.
- Yeniden kullanılabilir.
- EXEC veya SELECT ile çağrılır.
- Üç alt türü vardır 

Tür	Açıklama	Örnek
<b>Procedure</b>	Belirli bir işi yapan alt program	CREATE OR REPLACE PROCEDURE ...
<b>Function</b>	Bir değer döndüren alt program	CREATE OR REPLACE FUNCTION ...
<b>Trigger</b>	Olaylara tepki veren blok (ör. INSERT sonrası)	CREATE OR REPLACE TRIGGER ...
<b>Package</b>	Procedure ve function'ların gruplandığı yapı	CREATE OR REPLACE PACKAGE ...

# İsimli Blok - Örnek

```
CREATE OR REPLACE PROCEDURE selamla IS
BEGIN
    DBMS_OUTPUT.PUT_LINE('Merhaba, bu bir named block!');
END;
/
```

```
EXEC selamla;
```

Not: SQL Developer'da Run Script (F5) ile çalıştırırken / olmasa da çalışır.  
SQLPlus'ta / mutlaka olmalıdır.

# Exceptions - Örnek

```
BEGIN  
    DBMS_OUTPUT.PUT_LINE(10 / 0); -- Sıfıra bölme hatası  
EXCEPTION  
    WHEN ZERO_DIVIDE THEN  
        DBMS_OUTPUT.PUT_LINE('Sıfıra bölme hatası yakalandı!');  
END;  
/
```

Sıfıra bölme hatası yakalandı!

# Exceptions - Örnek

```
BEGIN  
    DBMS_OUTPUT.PUT_LINE(10 / NULL); -- NULL ile işlem hatası  
EXCEPTION  
    WHEN OTHERS THEN  
        DBMS_OUTPUT.PUT_LINE('Bilinmeyen bir hata oluştu.');//  
END;  
/
```

OTHERS → Tüm hataları yakalar.

# Exceptions – Örnek – Birden Fazla Hata Türü

```
DECLARE
    v_num NUMBER := 0;
BEGIN
    IF v_num = 0 THEN
        RAISE ZERO_DIVIDE; -- Manuel hata oluşturma
    END IF;
EXCEPTION
    WHEN ZERO_DIVIDE THEN
        DBMS_OUTPUT.PUT_LINE('Sıfıra bölme hatası yakalandı.');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Başka bir hata oluştu.');
END;
/
```

# Raise Exception ile Manuel Hata Oluşturma

```
DECLARE
    v_age NUMBER := -5;
BEGIN
    IF v_age < 0 THEN
        RAISE_APPLICATION_ERROR(-20001, 'Yaş negatif olamaz!');
    END IF;
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE(SQLERRM); -- Hata mesajını göster
END;
/
```

# Veri Tipleri

## Veri Tipi

**BOOLEAN**

**%TYPE**

**%ROWTYPE**

## Notlar

SQL'de yok, PL/SQL'de var → TRUE, FALSE, NULL değerleri alabilir.

Başka tablo veya değişkenin veri tipini referans almayı sağlar → Dinamik ve güvenli.

Bir tablodaki tüm sütunları tek değişkende toplamak için kullanılır.

```
DECLARE
    v_flag BOOLEAN;
BEGIN
    v_flag := TRUE;
    IF v_flag THEN
        DBMS_OUTPUT.PUT_LINE('Doğru');
    END IF;
END;
/
```

```
DECLARE
    v_emp employees%ROWTYPE;
BEGIN
    SELECT * INTO v_emp FROM employees WHERE employee_id = 100;
    DBMS_OUTPUT.PUT_LINE(v_emp.first_name || ' ' || v_emp.last_name);
END;
/
```

# PLSQL İçinde SQL Komutları Kullanımı

- DDL, DCL komutları PLSQL içinde doğrudan kullanılamaz. Execute ve Immediate komutları ile kullanılırlar.
- DML ve TCL komutları doğrudan kullanılabilirler.



# Static SQL - Select

- Static SQL = Kod yazılırken sorgu sabittir (compile time'da belli).
- PL/SQL bloğunda kullanılan klasik SELECT, INSERT, UPDATE, DELETE sorguları çoğunlukla static SQL'dir. SELECT için INTO kullanılmalıdır.
- Avantaj: Hata compile time'da anlaşılır, performansı genellikle iyidir.

```
DECLARE
    v_name employees.first_name%TYPE;
BEGIN
    SELECT first_name
    INTO v_name
    FROM employees
    WHERE employee_id = 100;

    DBMS_OUTPUT.PUT_LINE('Çalışan: ' || v_name);
END;
/
```

# Static SQL - Insert

- Static SQL = Kod yazılırken sorgu sabittir (compile time'da belli).
- PL/SQL bloğunda kullanılan klasik SELECT, INSERT, UPDATE, DELETE sorguları çoğunlukla static SQL'dir. SELECT için INTO kullanılmalıdır.
- Avantaj: Hata compile time'da anlaşılır, performansı genellikle iyidir.

```
BEGIN
    INSERT INTO employees_temp (employee_id, first_name,
last_name)
    VALUES (101, 'Ali', 'Yılmaz');

    DBMS_OUTPUT.PUT_LINE('Yeni kayıt eklendi.');
END;
/
```

# Static SQL - Update

- Static SQL = Kod yazılırken sorgu sabittir (compile time'da belli).
- PL/SQL bloğunda kullanılan klasik SELECT, INSERT, UPDATE, DELETE sorguları çoğunlukla static SQL'dir. SELECT için INTO kullanılmalıdır.
- Avantaj: Hata compile time'da anlaşılır, performansı genellikle iyidir.

```
BEGIN
    UPDATE employees_temp
    SET salary = salary * 1.1
    WHERE department_id = 10;

    DBMS_OUTPUT.PUT_LINE('Maaşlar güncellendi.');

END;
/
```

# Static SQL - Delete

- Static SQL = Kod yazılırken sorgu sabittir (compile time'da belli).
- PL/SQL bloğunda kullanılan klasik SELECT, INSERT, UPDATE, DELETE sorguları çoğunlukla static SQL'dir. SELECT için INTO kullanılmalıdır.
- Avantaj: Hata compile time'da anlaşılır, performansı genellikle iyidir.

```
BEGIN
    DELETE FROM employees_temp
    WHERE employee_id = 101;

    DBMS_OUTPUT.PUT_LINE('Kayıt silindi.');
END;
/
```

# Dynamic SQL - Select

- Dynamic SQL = Sorgu runtime'da oluşturulur, yani program çalışırken sorgu belirlenir.
- Avantaj: Tablo veya sütun isimleri değişken olabilir, koşullar runtime'a göre belirlenir.PL/SQL'de EXECUTE IMMEDIATE kullanılır.

```
DECLARE
    v_table_name VARCHAR2(30) := 'employees';
    v_sql        VARCHAR2(100);
    v_count      NUMBER;
BEGIN
    v_sql := 'SELECT COUNT(*) FROM ' || v_table_name;
    EXECUTE IMMEDIATE v_sql INTO v_count;

    DBMS_OUTPUT.PUT_LINE('Satır sayısı: ' || v_count);
END;
/
```

# Dynamic SQL - Insert

- Dynamic SQL = Sorgu runtime'da oluşturulur, yani program çalışırken sorgu belirlenir.
- Avantaj: Tablo veya sütun isimleri değişken olabilir, koşullar runtime'a göre belirlenir.PL/SQL'de EXECUTE IMMEDIATE kullanılır.

```
DECLARE
  v_table VARCHAR2(30) := 'employees_temp';
BEGIN
  EXECUTE IMMEDIATE 'INSERT INTO ' || v_table || '
(employee_id, first_name) VALUES (101, "Ali")';
  DBMS_OUTPUT.PUT_LINE('Yeni kayıt eklendi.');
END;
/
```

# Dynamic SQL - Update

- Dynamic SQL = Sorgu runtime'da oluşturulur, yani program çalışırken sorgu belirlenir.
- Avantaj: Tablo veya sütun isimleri değişken olabilir, koşullar runtime'a göre belirlenir.PL/SQL'de EXECUTE IMMEDIATE kullanılır.

```
DECLARE
  v_table VARCHAR2(30) := 'employees_temp';
BEGIN
  EXECUTE IMMEDIATE 'UPDATE ' || v_table ||
    ' SET first_name = "Ahmet" WHERE employee_id =
101';
  DBMS_OUTPUT.PUT_LINE('Kayıt güncellendi.');
END;
/
```

# Dynamic SQL - Delete

- Dynamic SQL = Sorgu runtime'da oluşturulur, yani program çalışırken sorgu belirlenir.
- Avantaj: Tablo veya sütun isimleri değişken olabilir, koşullar runtime'a göre belirlenir.PL/SQL'de EXECUTE IMMEDIATE kullanılır.

```
DECLARE
  v_table VARCHAR2(30) := 'employees_temp';
BEGIN
  EXECUTE IMMEDIATE 'DELETE FROM ' || v_table || ' WHERE
employee_id = 101';
  DBMS_OUTPUT.PUT_LINE('Kayıt silindi.');
END;
/
```

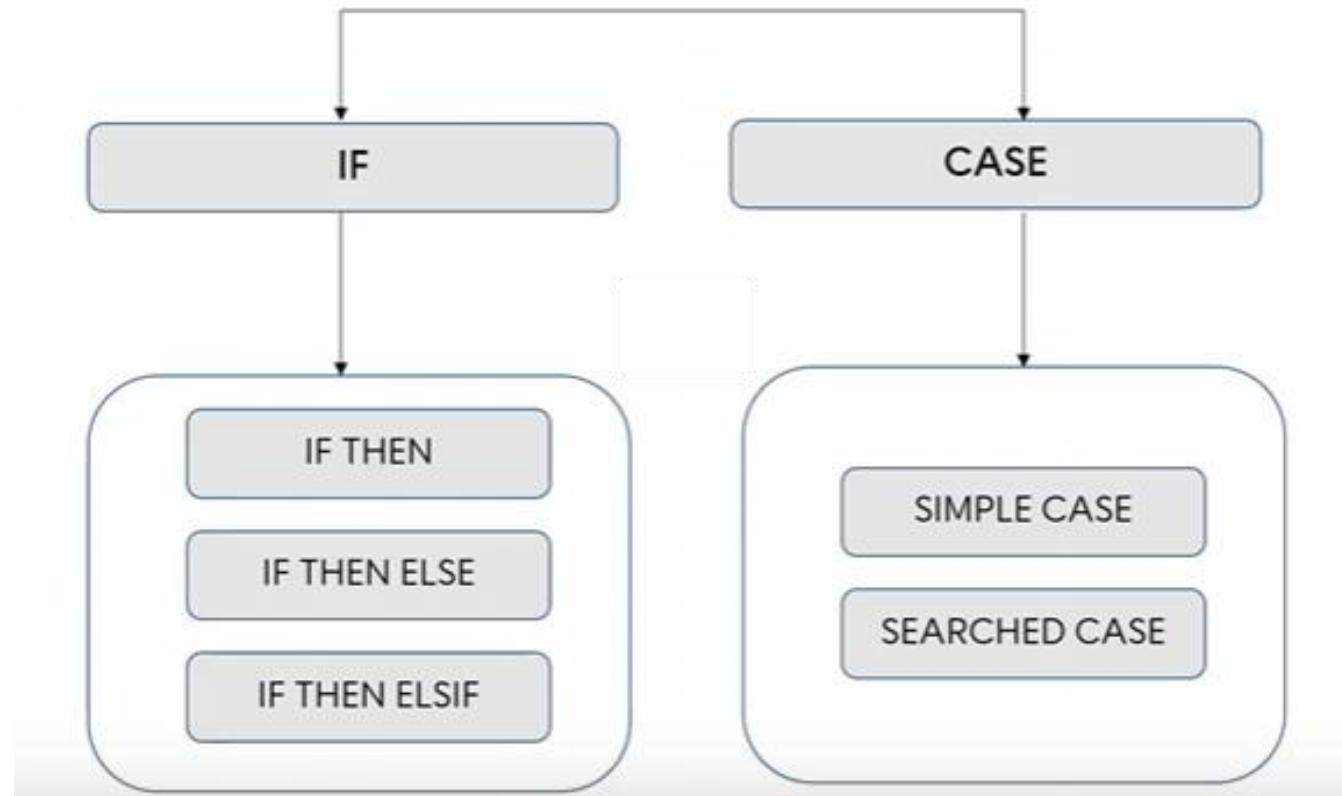
# Dynamic SQL – Grant, Revoke

- Dynamic SQL = Sorgu runtime'da oluşturulur, yani program çalışırken sorgu belirlenir.
- Avantaj: Tablo veya sütun isimleri değişken olabilir, koşullar runtime'a göre belirlenir. PL/SQL'de EXECUTE IMMEDIATE kullanılır.

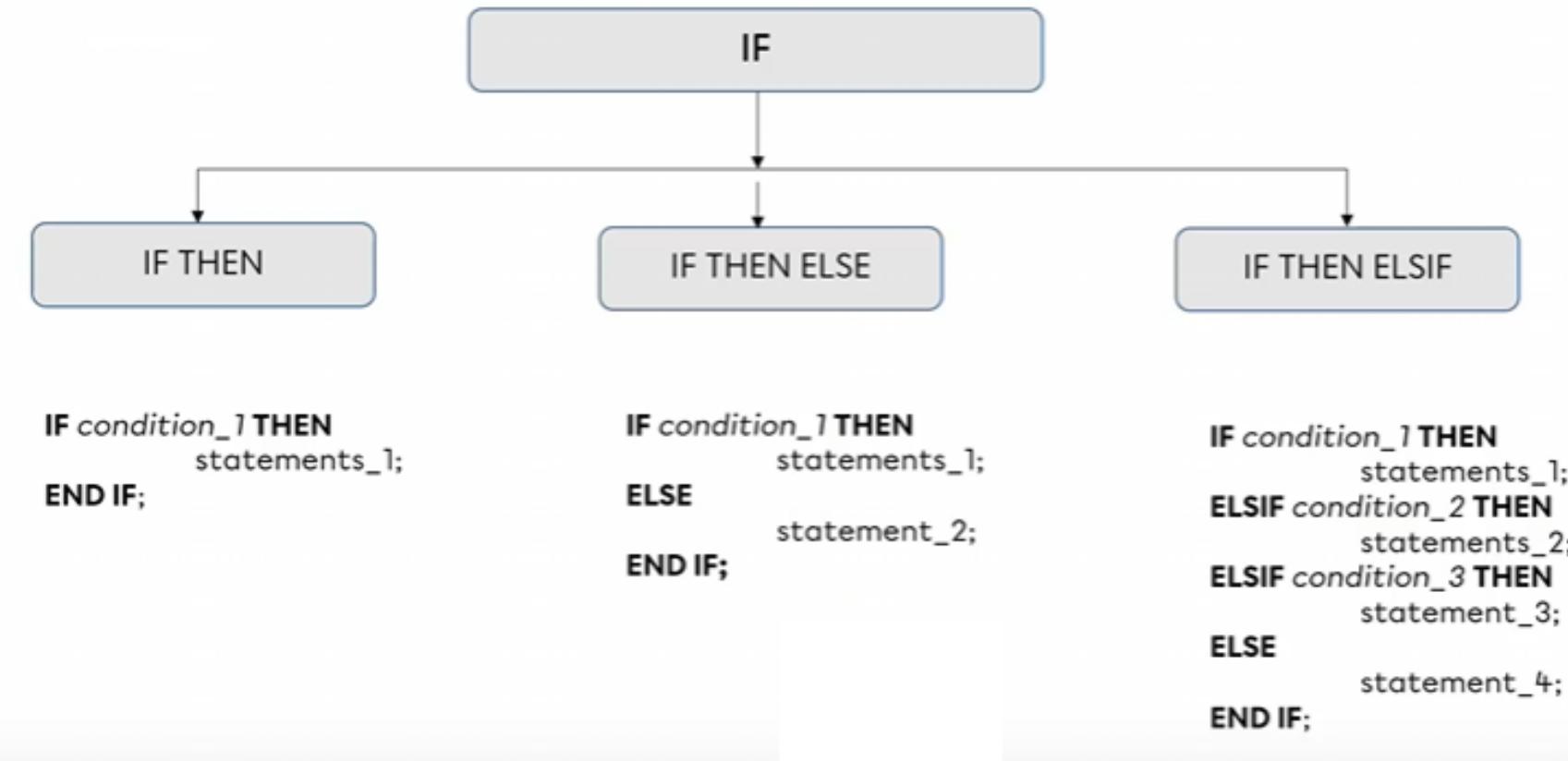
```
BEGIN  
  EXECUTE IMMEDIATE 'GRANT SELECT ON employees  
    TO EKREMM';  
  DBMS_OUTPUT.PUT_LINE('Yetki verildi.');//  
END;  
/
```

```
BEGIN  
  EXECUTE IMMEDIATE 'REVOKE SELECT ON employees FROM  
    SCOTT';  
  DBMS_OUTPUT.PUT_LINE('Yetki alındı.');//  
END;  
/
```

# Koşullu Akış Kontrolleri



# Koşullu Akış Kontrolleri - IF



# Koşullu Akış Kontrolleri – IF – IF THEN

```
DECLARE
    v_score NUMBER := 85;
BEGIN
    IF v_score >= 70 THEN
        DBMS_OUTPUT.PUT_LINE('Başarılı');
    END IF;
END;
/
```

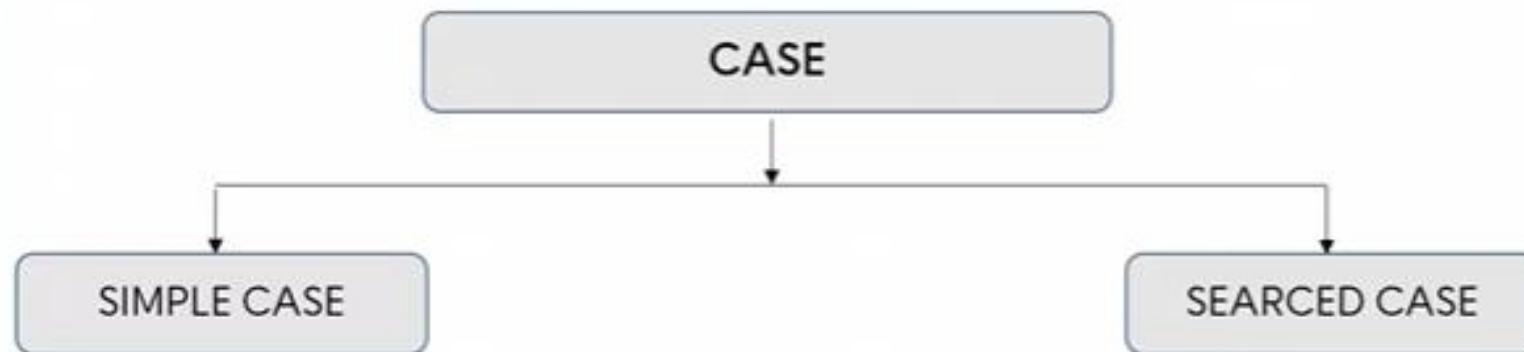
# Koşullu Akış Kontrolleri – IF – IF THEN ELSE

```
DECLARE
    v_score NUMBER := 55;
BEGIN
    IF v_score >= 70 THEN
        DBMS_OUTPUT.PUT_LINE('Başarılı');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Başarısız');
    END IF;
END;
/
```

# Koşullu Akış Kontrolleri – IF – IF THEN ELSIF

```
DECLARE
    v_score NUMBER := 75;
BEGIN
    IF v_score >= 90 THEN
        DBMS_OUTPUT.PUT_LINE('Mükemmel');
    ELSIF v_score >= 70 THEN
        DBMS_OUTPUT.PUT_LINE('İyi');
    ELSIF v_score >= 50 THEN
        DBMS_OUTPUT.PUT_LINE('Orta');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Başarısız');
    END IF;
END;
/
```

# Koşullu Akış Kontrolleri - CASE



**CASE selector**

```
WHEN selector_value_1 THEN statements_1  
WHEN selector_value_2 THEN statements_2  
...  
WHEN selector_value_n THEN statements_n  
[ ELSE else_statements ]  
END CASE;]
```

Tek bir ifade ile karşılaştırma yapılır (= value).

**CASE**

```
WHEN condition_1 THEN statements_1  
WHEN condition_2 THEN statements_2  
...  
WHEN condition_n THEN statements_n  
[ ELSE else_statements ]  
END CASE;]
```

Koşullar **boolean ifadeler** ile yapılır (>, <, BETWEEN, LIKE vb.)

# Koşullu Akış Kontrolleri – CASE – Simple CASE

```
DECLARE
    v_grade CHAR := 'B';
BEGIN
    DBMS_OUTPUT.PUT_LINE(
        CASE v_grade
            WHEN 'A' THEN 'Mükemmel'
            WHEN 'B' THEN 'İyi'
            WHEN 'C' THEN 'Orta'
            ELSE 'Başarısız'
        END
    );
END;
/
```

# Koşullu Akış Kontrolleri – CASE – Searched CASE

```
DECLARE
    v_score NUMBER := 75;
BEGIN
    DBMS_OUTPUT.PUT_LINE(
        CASE
            WHEN v_score >= 90 THEN 'Mükemmel'
            WHEN v_score >= 70 THEN 'İyi'
            WHEN v_score >= 50 THEN 'Orta'
            ELSE 'Başarisız'
        END
    );
END;
/
```