SQL NOTES

Distinct and unique Operator (synonym)
-They are used to eliminate the duplicate rows.

Select distinct job_id from employees;
select distinct first_name from employees;

They are some invalid usages of the DISTINCT operator.
The DISTINCT operator is a row-based operator. Only one DISTINCT operator is used in an SQL query.

Concatenation Operators || bunun ile;

WHERE — Filter data

<> both of them not equal
!=

```
Select first_name, last_name, salary
from employees
Where salary between 1000 and 12000;

Select first_name, last_name, salary
from employees
Where hire_date between '07-JUN-02' and '29-JAN-05';
```

IN

```
SELECT * FROM employees
WHERE employee_id IN (50, 100, 65, 210, 150);

SELECT * FROM employees
WHERE first_name IN ('Steven', 'Peter', 'Adam');

SELECT * FROM employees
WHERE first_name IN ('Steven', 'Peter', 'Adam', 'aa');

SELECT * FROM employees
WHERE hire_date IN ('08-MAR-08', '30-JAN-05');
```

LIKE

```
SELECT * FROM employees WHERE job_id = 'SA_REP';
SELECT * FROM employees WHERE job_id LIKE 'SA_REP';
SELECT * FROM employees WHERE job_id LIKE 'SA%';
SELECT * FROM employees WHERE first_name LIKE 'A%';
SELECT * FROM employees WHERE first_name LIKE '%a';
SELECT * FROM employees WHERE first_name LIKE '%a%';
SELECT * FROM employees WHERE first_name LIKE '_r%';
SELECT * FROM employees WHERE commission_pct = NULL;
SELECT * FROM employees WHERE commission_pct IS NULL;
SELECT * FROM employees WHERE commission_pct IS NOT NULL;
```

LOGICAL OPERATORS (AND, OR, NOT)

```
SELECT * FROM employees WHERE job_id = 'SA_REP' OR salary > 10000;
SELECT * FROM EMPLOYEES WHERE salary > 10000 AND job_id IN ('SA_MAN',
'SA_REP');

SELECT * FROM EMPLOYEES WHERE salary > 10000 AND job_id NOT IN ('SA_MAN',
'SA_REP');

SELECT first_name, last_name, job_id, salary FROM employees
WHERE (job_id = 'IT_PROG' or job_id = 'ST_CLERK') and salary > 5000;

SELECT first_name, last_name, job_id, salary FROM employees
WHERE job_id = 'IT_PROG' or (job_id = 'ST_CLERK' and salary > 5000);

SELECT first_name, last_name, job_id, salary FROM employees
WHERE job_id = 'IT_PROG' or job_id = 'ST_CLERK' and salary > 5000;

SELECT first_name, last_name, department_id, salary
FROM employees
WHERE salary > 10000 AND department_id = 20 OR department_id = 30;

SELECT first_name, last_name, department_id, salary
FROM employees
WHERE salary > 10000 AND (department_id = 20 OR department_id = 30);
```

!!! Önce AND daha sonra OR kısmına geçilir.

ORDER BY(ASC, DESC)

```
SELECT * FROM employees;

SELECT first_name, last_name, salary FROM employees ORDER BY last_name;

SELECT first_name, last_name, salary, (10*(salary/5) + 3000) - 100 NEW_SALARY
FROM employees ORDER BY NEW_SALARY;

SELECT first_name, last_name, salary, (10*(salary/5) + 3000) - 100 NEW_SALARY
FROM employees ORDER BY 1;
```

!!! Sorguda döndürülen sonuçları **ilk sütuna** göre sıralar.

```
SELECT first_name, last_name, salary, (10*(salary/5) + 3000) - 100 NEW_SALARY
FROM employees ORDER BY 2;
```

!!! ORDER BY 2 demek, **ikinci sütun (last_name)**'a göre sıralama yapılacağı anlamına gelir.

SELECT * FROM employees ORDER BY 2;

SELECT * FROM employees ORDER BY 5;

SELECT * FROM employees ORDER BY first_name, last_name;

!!! Çalışanları önce **adlarına** (first_name), sonra da **soyadlarına** (last_name) göre sıralar.

SELECT * FROM employees ORDER BY first_name, job_id, salary;

select employee_id, first_name, last_name, salary from employees order by first_name asc;

select employee_id, first_name, last_name, salary from employees order by first_name desc;

select employee_id, first_name, last_name, salary from employees order by first_name desc, last_name;

select employee_id, first_name, last_name, salary from employees order by first_name desc, last_name desc;

!!!ORDER BY first_name DESC: Çalışanlar önce **adlarına göre**, **Z'den A'ya** sıralanır.
ORDER BY last_name DESC: Aynı ada sahip çalışanlar varsa, bunlar **soyadlarına göre**, yine **Z'den A'ya** sıralanır.

select employee_id, first_name, last_name, salary from employees order by first_name desc, salary desc;

select employee_id, first_name, last_name, salary s from employees order by first_name desc, s desc;

select employee_id, first_name, last_name, salary s from employees order by 2 desc, s desc;

select first_name, salary, commission_pct from employees order by commission_pct;

NULLS FIRST , NULLS LAST

select first_name, salary, commission_pct from employees order by commission_pct;

select first_name, salary, commission_pct from employees order by commission_pct NULLS FIRST;

select first_name, salary, commission_pct from employees order by commission_pct ASC NULLS FIRST;

!!! Komisyon yüzdesi **NULL olmayanları** küçükten büyüğe sıralar, ancak **NULL değerli komisyon yüzdesine sahip çalışanlar** ilk sırada yer alır.

```
select first_name, salary, commission_pct from employees order by commission_pct
DESC;

select first_name, salary, commission_pct from employees order by commission_pct
DESC NULLS LAST;
```

FETCH

```
SELECT first_name, last_name, salary FROM employees
ORDER BY salary DESC
OFFSET 1 ROW FETCH FIRST 10 ROWS WITH TIES;
```

Çalışanları maaşlarına göre azalan sırayla sıralar.

!!!**İlk satırı atlar** (en yüksek maaşlı çalışanı dışarıda bırakır).
Sonra 10. sıradaki maaşa kadar olan ilk 10 satırı alır, ama maaşı 10. sıradaki çalışanla aynı olan başka çalışanlar varsa, bunlar da dahil edilir.

```
SELECT first_name, last_name, salary FROM employees
OFFSET 1 ROW FETCH FIRST 10 ROWS WITH TIES;
```

!!! İlk sıradaki çalışanı atlar (en yüksek maaşlıyı).
İkinci sıradaki çalışanla başlar ve sıralama yapılır.
İlk 10 maaşı alır ve maaşı 10. sıradaki ile aynı olanları da dahil eder.

```
SELECT first_name, last_name, salary FROM employees
ORDER BY salary DESC
FETCH FIRST 10 ROWS WITH TIES;
```

!!! Çalışanları **maaşlarına göre azalan sırayla** sıralar.
En yüksek maaşlı çalışanla başlar ve **ilk 10 maaşlı çalışanı** alır.
Maaşı 10. sıradaki çalışanla aynı olan çalışanlar da eklenir.

```
SELECT first_name, last_name, salary FROM employees
ORDER BY salary DESC
OFFSET 5 ROW;
```

!!! Çalışanları maaşlarına göre azalan sırayla sıralar, **ilk 5 çalışanı atlar** ve **6. sıradaki** çalışandan başlayarak verileri döndürür.

```
SELECT first_name, last_name, salary FROM employees
ORDER BY salary DESC
OFFSET 1 ROWS FETCH FIRST 10 ROWS WITH TIES;
```

ROWNUM, ROWID

```
SELECT employee_id, first_name, last_name, salary, rowid, rownum from employees;

SELECT employee_id, first_name, last_name, salary, rowid, rownum from employees
where department_id = 60;
```

```
SELECT employee_id, first_name, last_name, salary, rowid, rownum from employees
where department_id = 80;

SELECT employee_id, first_name, last_name, salary, rowid, rownum from employees
WHERE department_id = 80 and rownum <= 5 order by salary desc;
```

SUBSTITUONS VALUES

```
SELECT employee_id, first_name, last_name, department_id
FROM employees WHERE department_id = 30;

SELECT employee_id, first_name, last_name, department_id
FROM employees WHERE department_id = &department_no;
```

!!! Kullanıcıdan **department_no** adlı bir değişkenin (parametrenin) değerini alır.

```
SELECT employee_id, first_name, last_name, department_id
FROM employees WHERE first_name = '&name';
```

!!! Kullanıcıdan **name** adlı bir parametrenin değerini alır. Bu, çalışanların adlarına göre filtreleme yapar.

```
SELECT employee_id, first_name, last_name, department_id
FROM employees WHERE first_name = &name;
```

!!! Kullanıcıdan **name** adlı bir parametreyi alır. Ancak bu sorguda, **parametreyi tırnak içinde** almak yerine **direkt olarak girilmesi beklenir**.

```
SELECT employee_id, first_name, last_name, &column_name
FROM &table_name
WHERE &condition
ORDER BY &order_by_clause;
```

!!! Bu sorgu, kullanıcıdan alınan **dinamik parametrelerle** esnek bir şekilde çalışan bir sorgudur. Farklı tablo ve sütunlar üzerinde sorgulama yapma imkânı sağlar. Ancak, **güvenlik ve doğrulama** konularına dikkat edilmesi gereklidir.

DEFINE and UNDEFINE COMMANDS

```
SELECT employee_id, first_name, last_name, salary
FROM employees
WHERE salary BETWEEN &sal AND &sal + 1000;

SELECT employee_id, first_name, last_name, salary
FROM employees
WHERE salary BETWEEN &&sal AND &sal + 1000;

SELECT employee_id, first_name, last_name, &&column_name
FROM employees
ORDER BY &column_name;
```

```
SELECT &&column_name
FROM employees
GROUP BY &column_name
ORDER BY &column_name;

DEFINE emp_num = 100;
SELECT * FROM employees WHERE employee_id = &emp_num;
DEFINE emp_num = 200;
DEFINE column_name = 'first_name';
UNDEFINE emp_num;
DEFINE;
DEFINE column_name;
UNDEF column_name;
DEF column_name;
```

ACCEPT and PROMPT Commands

```
ACCEPT emp_id PROMPT 'Please Enter a valid Employee ID:';

SELECT employee_id, first_name, last_name, salary
FROM employees
WHERE employee_id = &emp_id;

UNDEFINE emp_id;

ACCEPT min_salary PROMPT 'Please specify the MINIMUM salary:'
ACCEPT max_salary PROMPT 'Please specify the MAXIMUM salary:'
SELECT employee_id, last_name, salary
FROM employees
WHERE salary BETWEEN &min_salary AND &max_salary;
UNDEFINE min_sal;
UNDEF max_sal;
```

İlk sorgu: Kullanıcıdan bir **çalışan ID'si** alır ve o ID'ye sahip çalışanın **ad, soyad ve maaş** bilgilerini getirir.
İkinci sorgu: Kullanıcıdan **minimum** ve **maksimum maaş** alır ve bu maaş aralığındaki çalışanların **ID, soyad ve maaş** bilgilerini döndürür.
UNDEFINE komutları, sorgulardan sonra alınan parametrelerin değerlerini **silerek** sonraki sorgularda kullanılmamalarını sağlar.

SET VERIFY ON - OFF Commands

```
SELECT employee_id, first_name, last_name, department_id
FROM employees WHERE department_id = &dept_id;

SET VERIFY ON;
SET VERIFY OFF;

SELECT * FROM departments WHERE department_name = 'R&D';
SET DEFINE OFF;
SET DEFINE ON;
```

&dept_id kullanılarak bir değişken (departman numarası) girilir ve çalışanlar bu departmanda sorgulanır.
SET VERIFY ON komutu, değişkenlerin yerine geçen değerlerin ekranda görünmesini sağlar.
SET VERIFY OFF komutu, değişkenlerin yerine geçen değerlerin ekranda görünmesini engeller.
SET DEFINE OFF komutu, SQL*Plus'ın değişkenleri tanımlama özelliğini kapatır.
SET DEFINE ON komutu, SQL*Plus'ın değişkenleri tanımlama özelliğini tekrar açar.

Case Conversion(LOWER, UPPER, INITCAP)

```
SELECT first_name, UPPER(first_name),
last_name, LOWER(last_name),
email, INITCAP(email) FROM employees;
```

!!! **INITCAP(email)**: **email** sütunundaki değeri **baş harfleri büyük**, diğer harfleri küçük yapacak şekilde dönüştürür. (Ancak, e-posta adresleri genellikle küçük harflerle yazılır, bu yüzden bu fonksiyon, e-posta adresinin baş harfini büyütmek gibi bir işlev görür.)

```
SELECT first_name, UPPER(first_name),
last_name, LOWER(last_name),
email, INITCAP(email) FROM employees
WHERE job_id = 'IT_PROG';

SELECT first_name, UPPER(first_name),
last_name, LOWER(last_name),
email, INITCAP(email),
UPPER('bmw i8')FROM employees
WHERE job_id = 'IT_PROG';

SELECT * FROM employees
WHERE last_name = 'KING';

SELECT * FROM employees
WHERE LOWER(last_name) = 'king';

SELECT * FROM employees
WHERE UPPER(last_name) = 'KING';

SELECT * FROM employees
WHERE INITCAP(last_name) = ‘King';
```

Character Manipulation Functions (Part 1)

```
SELECT first_name, SUBSTR(first_name,3,6), SUBSTR(first_name,3),
last_name, LENGTH(last_name)
FROM employees;
```

```
SELECT CONCAT(first_name, last_name)
FROM employees;

SELECT CONCAT(CONCAT(first_name, last_name),employee_id)
FROM employees;

SELECT first_name || last_name || employee_id
FROM employees;
```

INSTR(string, substring, start_position, occurrence)
INSTR() fonksiyonu, bir metin içinde belirli bir karakterin veya kelimenin **hangi konumda geçtiğini** bulur.

```
SELECT INSTR('I am learning how to use functions in Oracle', 'o', 17, 3) FROM dual;

SELECT INSTR('I am learning how to use functions in Oracle', 'o', 1, 3) FROM dual;
```

1 → Aramaya **baştan** (1. karakterden) başlanacak.
3 → **3. kez geçtiği** konum döndürülecek.

```
SELECT INSTR('I am learning how to use functions in Oracle', 'o', -1, 3) FROM dual;

SELECT INSTR('I am learning how to use functions in Oracle', 'o', -1, 1) FROM dual;

SELECT INSTR('I am learning how to use functions in Oracle', 'in', -1, 1) FROM dual;

SELECT INSTR('I am learning how to use functions in Oracle', 'in', 1, 1) FROM dual;

SELECT first_name,INSTR(first_name,'a') from employees;
```

TRIM, bir string'in başından (LEADING), sonundan (TRAILING) veya her iki tarafından (BOTH) belirli karakterleri kaldırmak için kullanılır.

```
SELECT TRIM (' My Name is Adam ') trm from dual;
SELECT TRIM (' ' FROM ' My Name is Adam ') trm from dual;
SELECT TRIM (BOTH ' ' FROM ' My Name is Adam ') trm from dual;
SELECT TRIM (LEADING ' ' FROM ' My Name is Adam ') trm from dual;
SELECT TRIM (TRAILING ' ' FROM ' My Name is Adam ') trm from dual;
SELECT TRIM (TRAILING 'm' FROM ' my Name is Adam ') trm from dual;
SELECT TRIM (TRAILING 'm' FROM 'my Name is Adam') trm from dual;
SELECT TRIM (TRAILING 'm' FROM 'my Name is Adammmmm') trm from dual;
SELECT TRIM (LEADING 'm' FROM 'my Name is Adam') trm from dual;
SELECT TRIM (BOTH 'm' FROM 'my Name is Adam') trm from dual;
SELECT TRIM ('m' FROM 'my Name is Adam') trm from dual;
```

RTRIM (Right Trim) - Sağdaki Belirtilen Karakterleri Kaldırır
RTRIM fonksiyonu, bir string'in sağındaki (sonundaki) boşlukları veya belirtilen karakterleri kaldırmak için kullanılır.

LTRIM (Left Trim) - Soldaki Belirtilen Karakterleri Kaldırır LTRIM fonksiyonu, bir string'in **solundaki** (başındaki) **boşlukları veya belirtilen karakterleri kaldırmak** için kullanılır.

```
SELECT RTRIM (' my Name is Adam ') trm from dual;
SELECT LTRIM (' my Name is Adam ') trm from dual;
SELECT LTRIM (' my Name is Adam ', 'my') trm from dual;
SELECT LTRIM ('my Name is Adam', 'my') trm from dual;
SELECT RTRIM ('my Name is Adam', 'my') trm from dual;
SELECT RTRIM ('my Name is Adammmm', 'my') trm from dual;
SELECT LTRIM ('www.mywebsite.com', 'w.') trm from dual;
SELECT LTRIM ('234234217www.mywebsite.com', '0123456789') trm from dual;
```

first_name sütunundaki her bir ismin içinde 'a' harfi varsa kaldırır.
Eğer 'a' harfi yerine başka bir karakter verilmezse, **sadece silme işlemi yapılır.**
Büyük-küçük harfe duyarlıdır, yani **'A' harflerini etkilemez.**

```
select first_name, replace(first_name,'a') rpl from employees;
select first_name, replace(first_name,'a','-') rpl from employees;
select first_name, replace(first_name,'le','-') rpl from employees;
select first_name, replace(first_name,'und','-') rpl from employees;
```

LPAD() XFonksiyonu (Left Padding - Sola Doldurma): Bir string'in sol tarafına belirli bir karakter ekleyerek belirtilen uzunluğa tamamlar.

RPAD() Fonksiyonu (Right Padding - Sağa Doldurma): Bir string'in sağ tarafına belirli bir karakter ekleyerek belirtilen uzunluğa tamamlar

```
select first_name, LPAD(first_name,10,'*') pad from employees;
select first_name, RPAD(first_name,10,'*') pad from employees;
select first_name, RPAD(first_name,6,'*') pad from employees;
select first_name, LPAD(first_name,6,'*') pad from employees;
select first_name, LPAD('My name is ',20,'-') pad from employees;
select first_name, LPAD('My name is '||last_name ,20,'-') pad from employees;
```

Character Manipulation Functions (Part 2) INSTR Function)

INSTR fonksiyonu, bir metin (string) içinde belirli bir karakter veya alt string'in (substring) kaçıncı konumda olduğunu bulur.
INSTR(string, substring [, start_position [, occurrence]])

- **string** → İçinde arama yapılacak metin.
- **substring** → Aranacak karakter veya kelime.
- **start_position** (Opsiyonel) → Aramaya başlanacak yer (**Pozitif: Baştan, Negatif: Sondan**).
- **occurrence** (Opsiyonel) → Kaçıncı eşleşme aranıyor (Varsayılan: 1).

```
SELECT INSTR('I am learning how to use functions in Oracle', 'o', 17, 3) FROM dual;
SELECT INSTR('I am learning how to use functions in Oracle', 'o', 1, 3) FROM dual;
SELECT INSTR('I am learning how to use functions in Oracle', 'o', -1, 3) FROM dual;
SELECT INSTR('I am learning how to use functions in Oracle', 'o', -1, 1) FROM dual;
SELECT INSTR('I am learning how to use functions in Oracle', 'in', -1, 1) FROM dual;
SELECT INSTR('I am learning how to use functions in Oracle', 'in', 1, 1) FROM dual;
SELECT first_name,INSTR(first_name,'a') from employees;
```

Character Functions - Part 4 (REPLACE, LPAD, RPAD Functions)

REPLACE() fonksiyonu, bir string içindeki belirli bir karakter veya kelimeyi **başka bir şeyle değiştirmek** için kullanılır.

```
SELECT first_name, REPLACE(first_name,'a') rpl FROM employees;
SELECT first_name, REPLACE(first_name,'a','-') rpl FROM employees;
SELECT first_name, REPLACE(first_name,'le','-') rpl FROM employees;
SELECT first_name, REPLACE(first_name,'und','-') rpl FROM employees;
```

LPAD() → Sola belirli karakter ekleyerek string'i belirtilen uzunluğa tamamlar.

RPAD() → Sağa belirli karakter ekleyerek string'i belirtilen uzunluğa tamamlar.

```
SELECT first_name, lpad(first_name,10,'*') pad FROM employees;
SELECT first_name, rpad(first_name,10,'*') pad FROM employees;
SELECT first_name, rpad(first_name,6,'*') pad FROM employees;
SELECT first_name, lpad(first_name,6,'*') pad FROM employees;
SELECT first_name, lpad('My name is ',20,'-') pad FROM employees;
SELECT first_name, lpad('My name is '||last_name ,20,'-') pad FROM employees;
```

Numeric Functions

- **ROUND()**, bir sayıyı belirli bir ondalık basamağa yuvarlar. Eğer ondalık basamak belirtilmezse, **en yakın tam sayıya yuvarlar**
- **TRUNC()**, sayıyı keserek belirli bir ondalık basamağa kadar gösterir (Yuvarlama yapmaz!).
- **CEIL()**, **ondalık kısmı ne olursa olsun** sayıyı **bir üst tam sayıya yuvarlar**.
- **FLOOR()**, ondalık kısmı ne olursa olsun **sayının en küçük tam sayıya yuvarlanmasını sağlar**.
- **MOD(a, b)** fonksiyonu, **"a" sayısının "b" sayısına bölümünden kalan değeri döndürür**.

```
SELECT round(12.536,2) FROM dual;
SELECT TRUNC(12.536,2) FROM dual;
SELECT ceil(12.536) FROM dual;
SELECT ceil(12.001) FROM dual;
SELECT ceil(12.999) FROM dual;
SELECT ceil(12) FROM dual;
SELECT floor(12.12) FROM dual;
SELECT floor(12.99) FROM dual;
SELECT MOD(8, 5) FROM dual;
SELECT MOD(8, 2) FROM dual;
SELECT MOD(1800, 70) FROM dual;
```

NESTED FUNCTIONS

```
SELECT SUBSTR('John Smith', INSTR('John Smith', ' ')+1,LENGTH('John Smith')) output
FROM dual;
```

INSTR('John Smith', ' ') → ' ' (boşluk) karakterinin konumunu bulur → **5**.
INSTR('John Smith', ' ')+1 → Boşluktan **sonraki karakterin** konumunu bulur → **6** (Yani 'S' harfi).
LENGTH('John Smith') → String'in uzunluğunu bulur → **10**.
SUBSTR('John Smith', 6, 10) → 6. karakterden başlayarak **10 karakter alır**.
Çıktı: 'Smith'.

```
SELECT SUBSTR('John Smith', INSTR('John Smith', ' ')+1) output
FROM dual;

SELECT first_name|| ' ' || last_name full_name,
SUBSTR(first_name|| ' ' || last_name,
INSTR(first_name|| ' ' || last_name, ' ')+1) output
FROM employees;

SELECT first_name|| ' ' || last_name full_name,
SUBSTR(concat(concat(first_name,' '),last_name),
INSTR(first_name|| ' ' || last_name, ' ')+1) output
FROM employees;
```

Date Functions & Arithmetic Operations on Dates

```
SELECT sysdate FROM dual;

SELECT sysdate, current_date, sessiontimezone, systimestamp, current_timestamp
FROM dual;

SELECT sysdate, sysdate + 4 FROM dual;
SELECT sysdate, sysdate - 4 FROM dual;
SELECT sysdate, sysdate + 1/24 FROM dual;
SELECT sysdate, sysdate + 1/(24*60) FROM dual;
SELECT employee_id, hire_date,sysdate FROM employees;

SELECT employee_id, hire_date,sysdate, sysdate-hire_date worked_in_days
FROM employees;

SELECT employee_id, hire_date,sysdate,trunc(sysdate-hire_date) worked_in_days
FROM employees;

SELECT employee_id, hire_date,sysdate,trunc((sysdate-hire_date)/365) worked_in_years
FROM employees
ORDER BY worked_in_years DESC;
```

Date Manipulation Functions in SQL

```
SELECT sysdate, add_months(sysdate,2) FROM dual;
SELECT sysdate, add_months(sysdate,-2) FROM dual;
SELECT sysdate, add_months(sysdate,30) FROM dual;
SELECT sysdate, add_months('12-JUL-21',30) FROM dual;

SELECT employee_id, hire_date,
trunc(hire_date,'MONTH') truncated_result, round(hire_date,'MONTH') rounded_result
FROM employees;

SELECT employee_id, hire_date,
trunc(hire_date,'YEAR') truncated_result, round(hire_date,'YEAR') rounded_result
FROM employees;


SELECT extract(year from sysdate) extracted_result FROM dual;
SELECT extract(month from sysdate) extracted_result FROM dual;
SELECT extract(day from sysdate) extracted_result FROM dual;
SELECT next_day(sysdate,'SUNDAY') next_day_result FROM dual;
This query finds the date of the next Sunday after the current date (sysdate).

SELECT last_day(sysdate) last_day_result FROM dual;
SELECT last_day('04-JUL-20') last_day_result FROM dual;
This query returns the last day of the month for the date '04-JUL-20'.
```

CONVERSION FUNCTIONS (TO_CHAR, TO_DATE, TO_NUMBER)

```
    --Implicit Conversion FROM a VARCHAR2 value TO a "NUMBER" value.
    SELECT * FROM EMPLOYEES WHERE salary > '5000';

    --Implicit Conversion FROM a VARCHAR2 value TO a "DATE" value.
    SELECT * FROM EMPLOYEES WHERE HIRE_DATE = '17-JUN-03';

    --Implicit Conversion FROM a NUMBER value TO a VARCHAR2 value.
    SELECT DEPARTMENT_ID || DEPARTMENT_NAME FROM DEPARTMENTS;

    --Implicit Conversion FROM a DATE value TO a VARCHAR2 value.
    SELECT FIRST_NAME || SYSDATE FROM EMPLOYEES;
```

TO_CHAR, TO_DATE, TO_NUMBER Functions (Part 1)

```
SELECT first_name, hire_date, to_char(hire_date,'YYYY') "Formatted Date" FROM
employees;

SELECT first_name, hire_date, to_char(hire_date,'YEAR') "Formatted Date" FROM
employees;

SELECT first_name, hire_date, to_char(hire_date,'MON-YYYY') "Formatted Date" FROM
employees;
```

TO_CHAR, TO_DATE, TO_NUMBER Functions (Part 2)

```
SELECT salary*commission_pct*100 "Original",
TO_CHAR(salary*commission_pct*100, '$999,999.00') "Formatted Version"
FROM employees WHERE commission_pct IS NOT NULL;

SELECT salary*commission_pct*100 "Original",
TO_CHAR(salary*commission_pct*100, 'L999,999.00') "Formatted Version"
FROM employees WHERE commission_pct IS NOT NULL;

SELECT salary*commission_pct*100 "Original",
TO_CHAR(salary*commission_pct*100, '$099,999.00') "Formatted Version"
FROM employees WHERE commission_pct IS NOT NULL;

SELECT to_number('$5,322.33', '$99,999.00') formatted_number FROM dual;
```

Null-Related (NVL, NVL2, NULLIF, COALESCE) Functions

```
SELECT employee_id, salary, commission_pct, salary + salary * commission_pct
FROM employees;

SELECT employee_id, salary, commission_pct, salary + salary * nvl(commission_pct,0)
nvl_new_sal
FROM employees;

SELECT first_name, last_name,
length(first_name) len1, length(first_name) len2,
nullif(length(first_name),length(last_name)) result
FROM employees;

SELECT coalesce(null,null,null,1,2,3, null) FROM dual;
SELECT coalesce(null,null,null,null) FROM dual;

SELECT state_province, city, coalesce(state_province,city) FROM locations;
```

CONDITIONAL EXPRESSIONS(CASE, DECODE)

```
-Example 1:
SELECT first_name, last_name, job_id, salary,
CASE job_id
        WHEN 'ST_CLERK' THEN salary * 1.2
        WHEN 'SA_REP' THEN salary * 1.3
        WHEN 'IT_PROG' THEN salary * 1.4
ELSE 0
END "UPDATED SALARY"
FROM employees;
```

```
--Example 2:
SELECT first_name, last_name, job_id, salary,
CASE job_id
        WHEN 'ST_CLERK' THEN salary * 1.2
        WHEN 'SA_REP' THEN salary * 1.3
        WHEN 'IT_PROG' THEN salary * 1.4
ELSE salary
END "UPDATED SALARY"
FROM employees;

--Example 3:
SELECT first_name, last_name, job_id, salary,
CASE
        WHEN job_id = 'ST_CLERK' THEN salary*1.2
        WHEN job_id = 'SA_REP' THEN salary*1.3
        WHEN job_id = 'IT_PROG' THEN salary*1.4
ELSE salary
END "UPDATED SALARY"
FROM employees;

--Example 4:
SELECT first_name, last_name, job_id, salary,
CASE
        WHEN job_id = 'ST_CLERK' THEN salary*1.2
        WHEN job_id = 'SA_REP' THEN salary*1.3
        WHEN job_id = 'IT_PROG' THEN salary*1.4
        WHEN last_name = 'King' THEN 2*salary
ELSE salary END "UPDATED SALARY"
FROM employees;

--Example 5:
SELECT first_name, last_name, job_id, salary,
CASE
        WHEN job_id = 'AD_PRES' THEN salary*1.2
        WHEN job_id = 'SA_REP' THEN salary*1.3
        WHEN job_id = 'IT_PROG' THEN salary*1.4
        WHEN last_name = 'King' THEN 2*salary
ELSE salary
END "UPDATED SALARY"
FROM employees;
--Example 6:
SELECT first_name, last_name, job_id, salary
FROM employees
WHERE (
CASE
        WHEN job_id = 'IT_PROG' AND salary > 5000 THEN 1
        WHEN job_id = 'SA_MAN' AND salary > 10000 THEN 1
ELSE 0
END) = 1;
```

Oracle Conditional Expressions DECODE Function

```
SELECT DECODE (1, 1,'One', 2,'Two') result FROM dual;

SELECT DECODE (25, 1,'One', 2,'Two',3,'Three','Not Found') result FROM dual;

SELECT first_name, last_name, job_id, salary,
DECODE(job_id,'ST_CLERK',salary*1.20,
'SA_REP' ,salary*1.30,
'IT_PROG' ,salary*1.50 ) as updated_salary
FROM EMPLOYEES;

SELECT first_name, last_name, job_id, salary,
DECODE(job_id,'ST_CLERK', salary*1.20,
'SA_REP' , salary*1.30,
'IT_PROG' , salary*1.50,
salary) as updated_salary
FROM EMPLOYEES;
```

GROUP FUNCTIONS(AVG, COUNT, MAX, MIN, SUM, LISTAGG)

```
SELECT avg(salary), avg(all salary), avg(distinct salary) FROM employees;

SELECT avg(salary), avg(all salary), avg(distinct salary), salary
FROM employees WHERE job_id = 'IT_PROG';

SELECT avg(commission_pct), avg(nvl(commission_pct,0)) FROM employees;

SELECT count(*), count(commission_pct), count(distinct commission_pct),
count(distinct nvl(commission_pct,0))
FROM employees;

SELECT min(salary), min(commission_pct), min(nvl(commission_pct,0)),
min(hire_date), min(first_name)
FROM employees

SELECT sum(salary), sum(ALL salary), sum(DISTINCT salary), sum(hire_date) FROM
employees;
```

LISTAGG Function

```
SELECT listagg(first_name,',') WITHIN GROUP (ORDER BY first_name) "Employees"
FROM employees
WHERE job_id = ‘ST_CLERK’;
```

Bu sorgu, **ST_CLERK** iş unvanına sahip çalışanların first_name (ilk ad) değerlerini alır ve virgülle ayırarak tek bir satırda birleştirir. ORDER BY first_name ifadesiyle ilk adlar alfabetik sıraya göre sıralanır.

```
SELECT listagg(first_name,',') WITHIN GROUP (ORDER BY last_name, salary DESC)
"Employees"
FROM employees
WHERE job_id = 'ST_CLERK';

SELECT listagg(salary,',') WITHIN GROUP (ORDER BY salary DESC) "Employees"
FROM employees
WHERE job_id = 'ST_CLERK';

SELECT listagg(distinct salary,' - ') WITHIN GROUP (ORDER BY salary DESC)
"Employees"
FROM employees
WHERE job_id = 'ST_CLERK';

SELECT * FROM locations;

SELECT listagg(city,',') WITHIN GROUP (ORDER BY city) AS CITIES
FROM locations
WHERE country_id = 'US';

SELECT listagg(city, ',') WITHIN GROUP (ORDER BY city) AS CITIES
FROM locations
WHERE country_id = ‘UK';
```

!!! İngiltere’deki şehirleri alfabetik sıraya göre sıralayarak virgülle ayırarak birleştirir.

```
SELECT listagg(city,',') AS CITIES
FROM locations
WHERE country_id = 'UK';

SELECT listagg(city) AS CITIES
FROM locations
WHERE country_id = 'UK';

SELECT j.job_title,
LISTAGG (e.first_name,', ') WITHIN GROUP (ORDER BY e.first_name) AS employees_list
FROM employees e, jobs j
WHERE e.job_id = j.job_id
GROUP BY j.job_title;

SELECT SUM(salary), AVG(salary), MAX(hire_date), MIN(commission_pct),
COUNT(DISTINCT manager_id), LISTAGG(job_id,',')
FROM employees;

SELECT SUM(salary), AVG(salary), MAX(hire_date), MIN(commission_pct),
COUNT(DISTINCT manager_id), LISTAGG(job_id,','), hire_date
FROM employees;
```

GROUPING DATA(GROUP BY, HAVING, NESTED GROUP FUNCTIONS)

GROUP BY Clause (Part 1)

```
SELECT avg(salary) FROM employees;

SELECT avg(salary) FROM employees WHERE job_id = 'IT_PROG';

SELECT avg(salary) FROM employees WHERE job_id = 'IT_PROG' or job_id = 'SA_REP';

SELECT job_id, avg(salary) FROM employees
GROUP BY job_id;

SELECT job_id, avg(salary) FROM employees
GROUP BY job_id
ORDER BY avg(salary) DESC;

SELECT job_id, department_id, manager_id, avg(salary), count(*) FROM employees
GROUP BY job_id, department_id, manager_id
ORDER BY count(*) DESC;

SELECT job_id, department_id, avg(salary), count(*) FROM employees
GROUP BY department_id, job_id, manager_id;

SELECT job_id, avg(salary) FROM employees
GROUP BY job_id;

SELECT job_id, avg(salary) FROM employees
GROUP BY job_id, department_id;

SELECT job_id, sum(salary), max(hire_date), count(*) FROM employees
GROUP BY job_id, department_id;

SELECT job_id, sum(salary), max(hire_date), count(*) FROM employees
WHERE job_id IN ('IT_PROG','ST_MAN','AC_ACCOUNT')
GROUP BY job_id;
```

HAVING Clause

```
SELECT job_id, avg(salary) FROM employees
WHERE avg(salary) > 10000
GROUP BY job_id;

SELECT job_id, avg(salary) FROM employees
GROUP BY job_id
HAVING avg(salary) > 10000;

SELECT job_id, avg(salary) FROM employees
WHERE hire_date > '28-MAY-05'
GROUP BY job_id
HAVING avg(salary) > 10000;
```

```
SELECT job_id, avg(salary) FROM employees
WHERE manager_id = 101
GROUP BY job_id
HAVING avg(salary) > 10000;

SELECT job_id, avg(salary) FROM employees
WHERE salary > 5000
GROUP BY job_id
--HAVING avg(salary) > 10000;

SELECT job_id, avg(salary) FROM employees
--WHERE salary > 10000
GROUP BY job_id
HAVING avg(salary) > 5000;
```

Nested Group Functions

```
SELECT department_id, max(avg(salary))
FROM employees
GROUP BY department_id;

SELECT max(avg(salary)), min(avg(salary))
FROM employees
GROUP BY department_id;
```

JOINS

Natural Join

```
DESC employees;
DESC departments;
SELECT * FROM employees;
SELECT * FROM departments;
SELECT * FROM employees NATURAL JOIN departments;
SELECT * FROM departments NATURAL JOIN employees;
SELECT first_name, last_name, department_name FROM departments NATURAL JOIN
employees;
```

Join with the USING Clause

```
SELECT * FROM employees NATURAL JOIN departments;

SELECT * FROM employees JOIN departments
USING(department_id);

SELECT * FROM employees JOIN departments
USING(department_id, manager_id);
```

Handling Ambiguous Column Names

```
SELECT * FROM employees JOIN departments
USING(department_id);

SELECT first_name, last_name, department_name, e.manager_id
FROM employees e J
OIN departments d
USING(department_id);

SELECT e.first_name, last_name, department_name, d.manager_id
FROM employees e
JOIN departments d
USING(department_id);

SELECT first_name, last_name, department_name, departments.manager_id
FROM employees e
JOIN departments
USING(department_id);

SELECT first_name, last_name, department_name, departments.manager_id
FROM employees e
JOIN departments
USING(manager_id);

SELECT first_name, last_name, department_name, manager_id
FROM employees e
JOIN departments
USING(manager_id);

SELECT first_name, last_name, department_name, manager_id
FROM employees e
JOIN departments
USING(e.manager_id);
```

Inner Join & Join with the ON Clause

```
SELECT e.first_name, e.last_name, d.manager_id, d.department_name
FROM employees e
JOIN departments d
ON(e.department_id = d.department_id AND e.manager_id = d.manager_id);

SELECT e.first_name, e.last_name, d.manager_id, d.department_name
FROM employees e
INNER JOIN departments d
ON(e.department_id = d.department_id AND e.manager_id = d.manager_id);

SELECT e.first_name, e.last_name, manager_id, d.department_name
FROM employees e
JOIN departments d
USING(department_id, manager_id);
```

```
SELECT e.first_name, e.last_name, d.manager_id, d.department_name
FROM employees e
JOIN departments d
ON(e.department_id = d.department_id AND e.employee_id = to_number(d.manager_id));
```

Multiple Join Operations

```
SELECT first_name, last_name, department_name, city, postal_code, street_address
FROM employees e
JOIN departments d
ON(e.department_id = d.department_id)
JOIN locations l
USING(location_id);
```

```
SELECT first_name, last_name, department_name, city, postal_code, street_address
FROM employees e
JOIN departments d
ON(e.department_id = d.department_id)
JOIN locations l
USING(location_id)
NATURAL JOIN COUNTRIES;
```

!!! **NATURAL JOIN** ifadesi, iki veya daha fazla tabloyu **ortak sütun adlarına** dayalı olarak birleştirir. Bu, JOIN işlemi yaparken hangi sütunların birleştirileceğini otomatik olarak belirler. Yani, NATURAL JOIN kullanıldığında, her iki tablodaki **aynı adı taşıyan sütunlar** birleştirilir.

```
SELECT first_name, last_name, department_name, city, postal_code, street_address,
country_id
FROM employees e
JOIN departments d
ON(e.department_id = d.department_id)
JOIN locations l
USING(location_id)
JOIN COUNTRIES
USING(country_id);
```

```
SELECT first_name, last_name, department_name, city, postal_code, street_address,
country_id
FROM employees e
JOIN departments d
ON(e.department_id = d.department_id)
JOIN COUNTRIES
USING(country_id)
JOIN locations l
USING(location_id);
```

Restricting Joins

```
SELECT first_name, last_name, department_name, city, postal_code, street_address
FROM employees e
JOIN departments d
ON(e.department_id = d.department_id)
JOIN locations l
ON(l.location_id = d.location_id)
WHERE e.job_id = 'IT_PROG';
```

```
SELECT first_name, last_name, department_name, city, postal_code, street_address
FROM employees e
JOIN departments d
ON(e.department_id = d.department_id)
JOIN locations l
ON(l.location_id = d.location_id)
AND e.job_id = 'IT_PROG'
AND e.first_name = ‘David’;
```

Self Join

```
SELECT worker.first_name, worker.last_name, worker.employee_id, worker.manager_id,
manager.employee_id, manager.first_name, manager.last_name, worker.salary,
manager.salary
FROM employees worker
JOIN employees manager
ON(worker.manager_id = manager.employee_id);
```

!!! **worker.manager_id = manager.employee_id** koşuluna göre, her çalışanın **manager_id**'si, ilgili yöneticinin **employee_id**'sine bağlanır. Bu, aynı tablonun birbirini referans alan iki ayrı kaydı arasında ilişki kurar.

Non-Equijoins (Joining Unequal Tables)

```
--Example 1:
SELECT e.employee_id, e.first_name, e.last_name, e.job_id, e.salary, j.min_salary,
j.max_salary, j.job_id
FROM employees e
JOIN jobs j
ON e.salary > j.max_salary
AND j.job_id = 'SA_REP';
```

```
--Example 2:
SELECT e1.employee_id, e1.first_name, e1.last_name
FROM employees e1
INNER JOIN employees e2
ON e1.employee_id <> e2.employee_id
AND e1.first_name = e2.first_name;
```

```
--Example 3:
SELECT e.first_name, e.last_name, j.job_title, e.salary, j.min_salary, j.max_salary
FROM employees e
JOIN jobs j
ON e.salary BETWEEN j.min_salary AND j.max_salary;
```

OUTER JOINS

```
SELECT first_name, last_name, department_name
FROM employees
JOIN departments
USING(department_id);

SELECT d.department_id, d.department_name, e.first_name, e.last_name
FROM departments d
JOIN employees e
ON (d.manager_id = e.employee_id);
```

LEFT OUTER JOIN (LEFT JOIN)

```
SELECT first_name, last_name, department_id, department_name
FROM employees
LEFT OUTER JOIN departments
USING(department_id);

SELECT e.first_name, e.last_name, d.department_id, d.department_name
FROM employees e
LEFT OUTER JOIN departments d
ON(e.department_id = d.department_id);

SELECT d.department_id, d.department_name, e.first_name, e.last_name
FROM departments d
LEFT JOIN employees e
ON(e.department_id = d.department_id);
```

!!! **İlk sorgu** tüm çalışanları gösterirken, sadece **eşleşen departman bilgilerini** döndürür.
İkinci sorgu ise tüm departmanları gösterir ve **eşleşen çalışan bilgilerini** döndürür.

RIGHT OUTER JOIN (RIGHT JOIN)

```
SELECT first_name, last_name, department_name
FROM employees e
RIGHT OUTER JOIN departments d
ON(e.department_id = d.department_id);

SELECT first_name, last_name, department_name, e.department_id, d.department_id
FROM employees e
RIGHT OUTER JOIN departments d
ON(e.department_id = d.department_id);
```

!!! **İlk sorgu** sadece çalışan adı ve soyadı ile departman adını döndürür ve eşleşmeyen departmanlar için çalışan bilgisi **NULL** olur.
İkinci sorgu ise her iki tablodan da **department_id**'yi gösterir. Çalışanlar tablosunda eşleşmeyen departmanlar için **çalışan bilgileri** NULL olur, ancak **departman bilgileri** her zaman gösterilir.

```
SELECT first_name, last_name, department_name, e.department_id, d.department_id
FROM employees e
LEFT OUTER JOIN departments d
ON(e.department_id = d.department_id);
```

```
SELECT first_name, last_name, department_name, e.department_id, d.department_id,
location_id
FROM employees e
RIGHT OUTER JOIN departments d
ON(e.department_id = d.department_id)
RIGHT OUTER JOIN locations
USING(location_id);
```

```
SELECT first_name, last_name, department_name, e.department_id, d.department_id,
location_id
FROM employees e RIGHT OUTER JOIN departments d
ON(e.department_id = d.department_id)
LEFT OUTER JOIN locations
USING(location_id);
```

Full Outer Join

```
SELECT first_name, last_name, department_name
FROM employees e
FULL OUTER JOIN departments d
ON(e.department_id = d.department_id);
```

```
SELECT first_name, last_name, department_name
FROM employees e
FULL JOIN departments d
ON(e.department_id = d.department_id);
```

!!! **departmanlar ile çalışanlar** arasında her iki tablodan da **eşleşmeyen kayıtları** gösterir. Her iki tabloyu **department_id**ile birleştirir ve eşleşmeyen kayıtlar için ilgili sütunları **NULL** yapar.

Cross Join (Cartesian Product / Cross Product)

```
SELECT first_name, last_name, department_name
FROM employees
CROSS JOIN departments;

SELECT first_name, last_name, department_name, job_title
FROM employees CROSS JOIN departments
CROSS JOIN jobs
WHERE job_title = 'Finance Manager';

SELECT d.department_name, j.job_title, COUNT(*) AS employee_count
FROM employees e JOIN departments d ON (e.department_id = d.department_id)
JOIN jobs j ON(j.job_id = e.job_id)
GROUP BY d.department_name, j.job_title
ORDER BY d.department_name, j.job_title;

SELECT c.department_name, c.job_title, COUNT(*) AS employee_count
FROM
        (SELECT d.department_name, j.job_title, j.job_id, d.department_id
        FROM departments d CROSS JOIN jobs j) c
LEFT OUTER JOIN employees e
ON (e.job_id = c.job_id AND e.department_id = c.department_id)
GROUP BY c.department_name, c.job_title
ORDER BY c.department_name, c.job_title;

SELECT c.department_name, c.job_title, COUNT(e.employee_id) AS employee_count
        FROM
        (SELECT d.department_name, j.job_title, j.job_id, d.department_id
        FROM departments d CROSS JOIN jobs j) c
LEFT OUTER JOIN employees e
ON (e.job_id = c.job_id AND e.department_id = c.department_id)
GROUP BY c.department_name, c.job_title
ORDER BY c.department_name, c.job_title;
```

The Differences Between The Inner Joins & Outer Joins, Equijoins & Non-Equijoins

```
---------- INNER EQUIJOIN EXAMPLES ----------
SELECT e.first_name, e.last_name, d.department_name
FROM employees e
JOIN departments d
ON (e.department_id = d.department_id);

SELECT e.first_name, e.last_name, d.department_name
FROM employees e
JOIN departments d
ON (e.department_id = d.department_id
AND e.manager_id = d.manager_id);
```

```
SELECT e.first_name, e.last_name, d.department_name, city
FROM employees e
JOIN departments d
ON (e.department_id = d.department_id)
JOIN locations l
ON (d.location_id = l.location_id);

SELECT e.first_name, e.last_name, d.department_name
FROM employees e, departments d
WHERE e.department_id = d.department_id;

---------- INNER NONEQUIJOIN EXAMPLES ----------
SELECT e.first_name, e.last_name, j.job_title, e.salary, j.min_salary, j.max_salary
FROM employees e
JOIN jobs j
ON (e.job_id = j.job_id AND e.salary > j.min_salary);

---------- OUTER EQIIJOIN EXAMPLES ----------
SELECT first_name, last_name, department_name, e.department_id emp_dept_id,
d.department_id dep_dept_id
FROM employees e
RIGHT OUTER JOIN departments d
ON(e.department_id = d.department_id);

SELECT first_name, last_name, department_name, e.department_id emp_dept_id,
d.department_id dep_dept_id
FROM employees e, departments d
WHERE e.department_id(+) = d.department_id;

SELECT first_name, last_name, department_name, e.department_id, d.department_id,
location_id
FROM employees e RIGHT OUTER JOIN departments d
ON(e.department_id = d.department_id)
RIGHT OUTER JOIN locations l
USING(location_id);

---------- OUTER NONEQUIJOIN EXAMPLES ----------
SELECT e.first_name, e.last_name, j.job_title, e.salary, j.min_salary, j.max_salary
FROM employees e LEFT OUTER JOIN jobs j
ON e.job_id = j.job_id
AND e.salary BETWEEN j.min_salary+500 AND j.max_salary;

SELECT e.first_name, e.last_name, j.job_title, e.salary, j.min_salary, j.max_salary
FROM employees e, jobs j
WHERE e.job_id = j.job_id(+)
AND e.salary BETWEEN j.min_salary(+)+500 AND j.max_salary(+);
```

SUBQUERIES

Using Subqueries

```
SELECT * FROM employees
WHERE salary > 14000;

SELECT * FROM employees
WHERE salary > (SELECT salary FROM employees
WHERE employee_id = 145);
```

Single Row Subqueries

```
--Step 1:----------------------------------------
SELECT * FROM employees;

--Step 2: Write the subquery first and enclose it with the parentheses.
(SELECT department_id FROM employees
WHERE employee_id = 145);

--Step 3: Use subquery/(ies) with the main query.
SELECT * FROM employees
WHERE department_id =
(SELECT department_id FROM employees
WHERE employee_id = 145)
AND salary <
(SELECT salary FROM EMPLOYEES
WHERE employee_id = 145);

--Step 5: The returning value from the subqueries must have the same data type with the
column you compared it to.
SELECT * FROM employees
WHERE employee_id =
(SELECT manager_id FROM employees
WHERE employee_id = 145)
AND salary <
(SELECT salary FROM EMPLOYEES
WHERE employee_id = 145);

SELECT * FROM employees
WHERE hire_date =
        (SELECT min(hire_date) FROM employees);

SELECT * FROM employees
WHERE hire_date =
                (SELECT max(hire_date) FROM employees
                GROUP BY department_id);
```

Multiple Row Subqueries

```
SELECT first_name, last_name, department_id, salary
FROM employees
WHERE salary IN (14000,15000,10000);

SELECT first_name, last_name, department_id, salary
FROM employees
WHERE salary IN (SELECT min(salary)
FROM employees
GROUP BY department_id);

SELECT first_name, last_name, department_id, salary
FROM employees
WHERE salary = ANY (SELECT salary
FROM employees
WHERE job_id = 'SA_MAN');

SELECT first_name, last_name, department_id, salary
FROM employees
WHERE salary < ANY (SELECT salary
FROM employees
WHERE job_id = 'SA_MAN');

SELECT first_name, last_name, department_id, salary
FROM employees
WHERE salary > ALL (SELECT salary
FROM employees
WHERE job_id = 'SA_MAN');

SELECT first_name, last_name, department_id, salary
FROM employees
WHERE department_id IN (SELECT department_id
FROM departments
WHERE location_id IN (SELECT location_id
FROM locations
WHERE country_id = (SELECT country_id
FROM countries
WHERE country_name = 'United Kingdom')));
```

Multiple Column Subqueries

—Nonpairwise

```
SELECT employee_id, first_name, last_name, department_id, salary
FROM employees
WHERE department_id IN
(SELECT department_id FROM employees
WHERE employee_id IN (103,105,110))
AND salary IN
(SELECT salary FROM employees
WHERE employee_id IN (103,105,110));
```

```
--Pairwise
SELECT employee_id, first_name, last_name, department_id, salary
FROM employees
WHERE (department_id,salary) IN
(SELECT department_id, salary
FROM employees
WHERE employee_id IN (103,105,110));

--Nonpairwise Ex2
SELECT employee_id, first_name, last_name, department_id, salary
FROM employees
WHERE department_id IN
(SELECT department_id FROM employees)
AND salary IN
(SELECT max(salary) FROM employees
GROUP BY department_id);
```

Using Subqueries as a Table

```
SELECT *
FROM (SELECT department_id, department_name, state_province, city
FROM departments JOIN locations
USING (location_id)
ORDER BY department_id);

SELECT manager_id
FROM (SELECT department_id, department_name, state_province, city
FROM departments JOIN locations
USING (location_id)
ORDER BY department_id);

SELECT e.employee_id, e.first_name, e.last_name, b.department_name, b.city,
b.state_province
FROM employees e JOIN (SELECT department_id, department_name, state_province,
city
FROM departments JOIN locations
USING (location_id)
ORDER BY department_id) b
USING (department_id);
```

SCALAR Subqueries

```
SELECT first_name, last_name, job_id
FROM employees
WHERE salary > (SELECT avg(salary) FROM employees);

SELECT * FROM employees
WHERE department_id = (SELECT department_id FROM employees
WHERE first_name = 'Luis');
```

```
SELECT employee_id, first_name, last_name,
(       CASE
        WHEN location_id = (SELECT location_id FROM locations WHERE
postal_code = '99236')
        THEN 'San Francisco'
        ELSE 'Other'
        END) city
FROM employees NATURAL JOIN departments;

SELECT * FROM employees
WHERE department_id = (SELECT department_id FROM employees
WHERE first_name = 'Luisee');

SELECT *
FROM employees
WHERE (department_id, manager_id) = (SELECT department_id, manager_id
FROM employees
WHERE first_name ='Luis');
```

CORRELATED Subqueries

```
--Example 1:
SELECT employee_id, first_name, last_name, department_id, salary
FROM employees a
WHERE salary = (SELECT max(salary)
FROM employees b
WHERE b.department_id = a.department_id);

SELECT employee_id, first_name, last_name, department_id, salary
FROM employees a
WHERE (salary, department_id) IN (SELECT max(salary), department_id
FROM employees b
GROUP BY department_id);

SELECT employee_id, first_name, last_name, a.department_id, salary
FROM employees a JOIN (SELECT avg(salary) avg_sal, department_id
FROM employees
GROUP BY department_id) b
ON a.department_id = b.department_id
WHERE a.salary < b.avg_sal;

--Example 2:
SELECT employee_id, first_name, last_name, department_name, salary,
(SELECT round(avg(salary))
FROM employees
WHERE department_id = d.department_id) "DEPARMENT'S AVERAGE SALARY"
FROM employees e JOIN departments d
ON (d.department_id = e.department_id)
ORDER BY e.employee_id;
```

EXISTS Operator & Semijoins

```
SELECT employee_id, first_name, last_name, department_id, salary
FROM employees a
WHERE EXISTS
        (SELECT * FROM employees WHERE manager_id = a.employee_id);

SELECT employee_id, first_name, last_name, department_id, salary
FROM employees a
WHERE EXISTS
        (SELECT commission_pct*10 FROM employees WHERE manager_id =
a.employee_id);
```

NOT EXISTS Operator

```
SELECT * FROM departments d
WHERE NOT EXISTS
        (SELECT department_id FROM employees e
        WHERE e.department_id = d.department_id);

SELECT * FROM departments d
WHERE NOT EXISTS
        (SELECT null FROM employees e
        WHERE e.department_id = d.department_id);

SELECT *
        FROM departments d
        WHERE department_id NOT IN
        (SELECT department_id FROM employees);
```

UNION, UNION ALL, MINUS, ORDER BY, SET

UNION and UNION ALL Operators

```
SELECT * FROM retired_employees
UNION
SELECT * FROM employees;

SELECT * FROM retired_employees
UNION
SELECT * FROM employees WHERE job_id = 'IT_PROG';

SELECT first_name, last_name, email, hire_date, salary FROM retired_employees
UNION
SELECT first_name, last_name, email, hire_date, department_id FROM employees;
```

UNION operatörü kullanarak **retired_employees** (emekli olmuş çalışanlar) ve **employees** (aktif çalışanlar) tablolarındaki verileri birleştirir. **UNION** operatörü, iki sorgudan dönen sonuçları birleştirir ve **tekrarlanan satırları** otomatik olarak kaldırır.

```
SELECT first_name, last_name, email, hire_date, salary, job_id FROM retired_employees
UNION ALL
SELECT first_name, last_name, email, hire_date, salary, job_id FROM employees;
```

INTERSECT Operator

```
SELECT first_name, last_name, email, hire_date, salary, job_id FROM retired_employees
INTERSECT
SELECT first_name, last_name, email, hire_date, salary, job_id FROM employees;
```

MINUS Operator

```
select first_name, last_name, email, hire_date, salary, job_id from retired_employees
minus
select first_name, last_name, email, hire_date, salary, job_id from employees;

select first_name, last_name, email, hire_date, job_id from retired_employees
minus
select first_name, last_name, email, hire_date, job_id from employees;

select first_name, last_name, email, hire_date, salary, job_id from employees
minus
select first_name, last_name, email, hire_date, salary, job_id from retired_employees;

select first_name from employees
minus
select first_name from retired_employees;
```

Matching Unmatched Queries in SET Operations

```
SELECT job_id, department_id, first_name, last_name FROM employees
UNION ALL
SELECT job_id, department_id, NULL, NULL FROM job_history;

SELECT job_id, NULL department_id, first_name, last_name FROM employees
UNION ALL
SELECT job_id, department_id, NULL, NULL FROM job_history;

SELECT job_id, 0 department_id, first_name, last_name FROM employees
UNION ALL
SELECT job_id, department_id, NULL, NULL FROM job_history;
```

Using the ORDER BY Clause with SET Operators

```
SELECT first_name, last_name, salary, department_id FROM employees
UNION
SELECT first_name, last_name, salary, department_id FROM retired_employees
ORDER BY salary;
```

```
SELECT first_name, last_name, salary, department_id FROM employees
UNION ALL
SELECT first_name, last_name, salary s, department_id FROM employees WHERE
department_id = 30
UNION
SELECT first_name, last_name, salary, department_id FROM retired_employees
ORDER BY salary DESC;

SELECT first_name, last_name, salary s, department_id FROM employees
UNION ALL
SELECT first_name, last_name, salary s, department_id FROM employees WHERE
department_id = 30
UNION
SELECT first_name, last_name, salary, department_id FROM retired_employees;
```

Combining Multiple Queries Using the SET Operators

```
SELECT first_name, last_name, salary, department_id FROM employees
UNION ALL
SELECT first_name, last_name, salary, department_id FROM employees WHERE
department_id = 30
UNION
SELECT first_name, last_name, salary, department_id FROM retired_employees
ORDER BY salary;

SELECT first_name, last_name, salary, department_id FROM employees
UNION ALL
SELECT first_name, last_name, salary, department_id FROM employees WHERE
department_id = 30
MINUS
SELECT first_name, last_name, salary, department_id FROM retired_employees
ORDER BY salary;

SELECT first_name, last_name, salary, department_id FROM employees
MINUS
SELECT first_name, last_name, salary, department_id FROM employees WHERE
department_id = 30
INTERSECT
SELECT first_name, last_name, salary, department_id FROM retired_employees
ORDER BY salary;
```

DLL —> READ ONLY, DROP TABLE, ALTER TABLE, TRUNCATE, COMMENT, RENAME

CREATE TABLE Statement

```
DESC employees;
CREATE TABLE my_employees(employee_id NUMBER(3) NOT NULL,
                          first_name VARCHAR2(50) DEFAULT 'No Name',
                          last_name VARCHAR2(50),
                          hire_date DATE DEFAULT
sysdate NOT NULL);

SELECT * FROM my_employees;

INFO my_employees;

CREATE TABLE my_employees(employee_id NUMBER(3) NOT NULL,
                          first_name VARCHAR2(50) DEFAULT 'No Name',
                          last_name VARCHAR2(50),
                          hire_date DATE DEFAULT sysdate NOT NULL,
                          email VARCHAR2(20));
```

CREATE TABLE AS SELECT Statement

```
SELECT * FROM employees WHERE 1=2;

CREATE TABLE employees_copy AS SELECT * FROM employees;
CREATE TABLE employees_copy2 AS SELECT * FROM employees;
SELECT * FROM employees_copy2;

CREATE TABLE employees_copy3 AS
SELECT * FROM employees WHERE 1=2;
SELECT * FROM employees_copy3;

CREATE TABLE employees_copy4 AS
SELECT * FROM employees WHERE job_id = 'IT_PROG';
SELECT * FROM employees_copy4;

CREATE TABLE employees_copy5 AS
SELECT first_name, last_name, salary FROM employees;
SELECT * FROM employees_copy5;

CREATE TABLE employees_copy6 AS
SELECT first_name, last_name l_name, salary FROM employees;
SELECT * FROM employees_copy6;

CREATE TABLE employees_copy7 (name, surname) AS
SELECT first_name, last_name l_name, salary FROM employees;
CREATE TABLE employees_copy7 (name, surname, annual_salary) AS
SELECT first_name, last_name l_name, salary*12 FROM employees;
SELECT * FROM employees_copy7;
DESC employees_copy7;
```

ALTER TABLE Statement

```
CREATE TABLE my_employees (employee_id NUMBER(3),
                           first_name VARCHAR2(50),
                           hire_date DATE DEFAULT sysdate);

CREATE TABLE my_employees (employee_id NUMBER(3),
                           first_name VARCHAR2(50),
                           hire_date DATE DEFAULT sysdate,
                           phoneVARCHAR2(20));

ALTER TABLE employees_copy ADD ssn varchar2(11);

SELECT * FROM employees_copy;

ALTER TABLE employees_copy
ADD (fax_number VARCHAR2(11),
     birth_date DATE,
     password VARCHAR2(10) DEFAULT 'abc1234');

ALTER TABLE employees_copy MODIFY passwordd VARCHAR2(50);

ALTER TABLE employees_copy MODIFY (fax_number VARCHAR2(11) DEFAULT '-',
                                   password VARCHAR2(10));

ALTER TABLE employees_copy MODIFY (fax_number VARCHAR2(11) DEFAULT NULL,
                                   password VARCHAR2(10) NOT NULL);

ALTER TABLE employees_copy MODIFY (fax_number VARCHAR2(11) DEFAULT NULL,
                                   password VARCHAR2(10) DEFAULT '0000');

ALTER TABLE employees_copy DROP COLUMN ssn;

ALTER TABLE employees_copy DROP (fax_number, password);

ALTER TABLE employees_copy DROP (birth_date);
```

Marking Columns Unused (SET UNUSED Clause)

```
ALTER TABLE employees_copy SET UNUSED (first_name, phone_number, salary);
DESC employees_copy;

SELECT * FROM USER_UNUSED_COL_TABS;

ALTER TABLE employees_copy SET UNUSED COLUMN last_name ONLINE;

ALTER TABLE employees_copy DROP UNUSED COLUMNS;
```

READ-ONLY Tables in SQL

```
CREATE TABLE emp_temp AS SELECT * FROM employees;

SELECT * FROM emp_temp;

ALTER TABLE emp_temp READ ONLY;

DELETE emp_temp;

ALTER TABLE emp_temp ADD gender VARCHAR2(1);

ALTER TABLE emp_temp DROP (gender);

DROP TABLE emp_temp;

ALTER TABLE emp_temp READ WRITE;
```

DROP TABLE Statement

```
SELECT * FROM employees_copy6;

DROP TABLE employees_copy6;
DROP TABLE employees_copy3, employees_copy4;
DROP employees_copy3, employees_copy4;
DROP TABLES employees_copy3, employees_copy4;

SELECT * FROM employees_copy4;

DROP TABLE employees_copy4;
FLASHBACK TABLE employees_copy4 TO BEFORE DROP;
DROP TABLE employees_copy4 PURGE;
```

TRUNCATE TABLE Statement

```
SELECT * FROM employees_copy;
DELETE FROM employees_copy;
TRUNCATE TABLE employees_copy;
DROP TABLE employees_copy;

CREATE TABLE employees_performance_test AS SELECT e1.first_name, e1.last_name,
e1.department_id, e1.salary
FROM employees e1 CROSS JOIN employees e2 CROSS JOIN employees e3;

SELECT COUNT(*) FROM employees_performance_test;
DELETE FROM employees_performance_test;
TRUNCATE TABLE employees_performance_test;
DROP TABLE employees_performance_test;
```

COMMENT Statement

```
CREATE TABLE employees_copy AS SELECT * FROM employees;

COMMENT ON COLUMN employees_copy.job_id IS 'Stores job title abbreviations';
COMMENT ON TABLE employees_copy IS 'This is a copy of the employees table’;
COMMENT ON COLUMN employees_copy.hire_date IS 'The date when the employee
started this job';
COMMENT ON COLUMN employees_copy.hire_date IS '';
COMMENT ON COLUMN employees_copy.hire_date IS ‘This is a sample comment’;

SELECT * FROM user_tab_comments;

SELECT * FROM user_tab_comments WHERE table_name = 'EMPLOYEES_COPY';

SELECT * FROM user_col_comments WHERE table_name = ‘EMPLOYEES_COPY’;
```

RENAME Statement

```
DESC employees_copy;
ALTER TABLE employees_copy RENAME COLUMN hire_date TO start_date;

RENAME employees_copy TO employees_backup;

SELECT * FROM employees_copy;
SELECT * FROM employees_backup;

ALTER TABLE employees_backup RENAME TO employees_copy;
SELECT * FROM employees_copy;
INSERT Statement (Part 1)
CREATE TABLE jobs_copy AS SELECT * FROM jobs;
DESC jobs_copy;
```

DML —> INSERT, UPDATE, DELETE, MERGE, SAVEPOINT, FOR UPDATE

```
INSERT INTO jobs_copy (job_id, job_title, min_salary, max_salary)
VALUES('GR_LDR', 'Group Leader', 8500, 20000);
SELECT * FROM jobs_copy;

INSERT INTO jobs_copy (job_id, job_title, min_salary, max_salary)
VALUES('PR_MGR', 'Project Manager', 7000, 18000);

INSERT INTO jobs_copy (job_title, min_salary, job_id, max_salary)
VALUES('Architect',6500,'ARCH',15000);

INSERT INTO jobs_copy
VALUES('DATA_ENG','Data Engineer',8000,21000);

INSERT INTO jobs_copy (job_id, job_title, min_salary)
VALUES('DATA_ARCH','Data Architecture',8000);

ALTER TABLE jobs_copy MODIFY max_salary DEFAULT 10000;
```

```
INSERT INTO jobs_copy (job_id, job_title, min_salary)
VALUES('DATA_ARCH2','Data Architecture2',8000);

INSERT INTO jobs_copy (job_id, min_salary)
VALUES(‘DATA_ARCH2',8000);
```

INSERT Statement (Part 2)

```
INSERT INTO jobs_copy
VALUES('DATA_ARCH2',8000);

INSERT INTO jobs_copy
VALUES('DATA_ARCH2','Data Architecture2',8000);

INSERT INTO jobs_copy
VALUES('DATA_ARCH3','Data Architecture3',8000, NULL);

SELECT * FROM employees_copy;

INSERT INTO employees_copy SELECT * FROM employees;

INSERT INTO employees_copy SELECT * FROM employees WHERE job_id ='IT_PROG';

INSERT INTO employees_copy(first_name,last_name,email,hire_date,job_id)
SELECT first_name,last_name,email,hire_date,job_id
FROM employees
WHERE job_id = 'IT_PROG';

INSERT INTO employee_addresses
SELECT employee_id, first_name, last_name, city || ' - ' || street_address AS address
FROM employees
JOIN departments USING (department_id)
JOIN locations USING (location_id);

CREATE TABLE employee_addresses AS
SELECT employee_id, first_name, last_name, city || ' - ' || street_address AS address
FROM employees
JOIN departments USING (department_id)
JOIN locations USING (location_id)
WHERE 1=2;

SELECT * FROM employee_addresses;
```

Unconditional Insert Statements (INSERT ALL Statements)

```
--Creates the employees_history table with no data based on the employees table
CREATE TABLE employees_history AS
SELECT employee_id, first_name, last_name, hire_date
FROM employees WHERE 1=2;
```

```
--Creates the salary_history table with no data based on the employees table
--1234 and 12 are just ordinary numbers, to make the data type of these columns number
CREATE TABLE salary_history AS
SELECT employee_id, 1234 AS year, 12 AS month, salary, commission_pct
FROM employees WHERE 1=2;

--Inserts all the returning rows into two different tables in one step
INSERT ALL
INTO employees_history VALUES (employee_id,first_name,last_name,hire_date)
INTO salary_history VALUES (employee_id, EXTRACT(year FROM sysdate),
EXTRACT(month FROM sysdate),salary, commission_pct)
SELECT * FROM employees WHERE hire_date> TO_DATE('15-MAR-08');

SELECT * FROM employees_history;

SELECT * FROM salary_history;

--Insert multiple rows into the same table, with static values
INSERT ALL
INTO employees_history VALUES (105,'Adam','Smith',sysdate)
INTO employees_history VALUES (106,'Paul','Smith',sysdate+1)
SELECT * FROM dual;
```

Conditional INSERT ALL Statements

```
--Creates a table called it_programmers with no data based on the employees table
columns
CREATE TABLE it_programmers AS
SELECT employee_id, first_name, last_name, hire_date FROM employees WHERE 1=2;

--Creates a table called working_in_us with no data based on the employees table
columns
CREATE TABLE working_in_the_us AS
SELECT employee_id, first_name, last_name, job_id, department_id FROM employees
WHERE 1=2;

INSERT ALL
WHEN hire_date > to_date('15-MAR-08') THEN
INTO employees_history VALUES (employee_id,first_name,last_name,hire_date)
INTO salary_history VALUES (employee_id, EXTRACT(year FROM sysdate),
EXTRACT(month FROM sysdate),salary, commission_pct)
WHEN job_id = 'IT_PROG' THEN
INTO it_programmers VALUES(employee_id,first_name,last_name,hire_date)
WHEN department_id IN
(SELECT department_id FROM departments WHERE location_id IN
(SELECT location_id FROM locations WHERE country_id = 'US')) THEN
INTO working_in_the_us VALUES
(employee_id,first_name,last_name,job_id,department_id)
```

```
SELECT * FROM employees;
SELECT * FROM it_programmers;
SELECT * FROM working_in_the_us;
SELECT * FROM employees_history;

INSERT ALL
WHEN hire_date > to_date('15-MAR-08') THEN
INTO salary_history VALUES (employee_id, EXTRACT(year FROM sysdate),
EXTRACT(month FROM sysdate),salary, commission_pct)
WHEN job_id = 'IT_PROG' THEN
INTO it_programmers VALUES(employee_id,first_name,last_name,hire_date)
WHEN department_id IN
(SELECT department_id FROM departments WHERE location_id IN
(SELECT location_id FROM locations WHERE country_id = 'US')) THEN
INTO working_in_the_us VALUES
(employee_id,first_name,last_name,job_id,department_id)
ELSE
INTO employees_history VALUES (employee_id,first_name,last_name,hire_date)
SELECT * FROM employees;

INSERT ALL
WHEN hire_date > to_date('15-MAR-08') THEN
INTO salary_history VALUES (employee_id, EXTRACT(year FROM sysdate),
EXTRACT(month FROM sysdate),salary, commission_pct)
WHEN 1=1 THEN
INTO it_programmers VALUES(employee_id,first_name,last_name,hire_date)
WHEN department_id IN
(SELECT department_id FROM departments WHERE location_id IN
(SELECT location_id FROM locations WHERE country_id = 'US')) THEN
INTO working _in_the_us VALUES
(employee_id,first_name,last_name,job_id,department_id)
ELSE
INTO employees_history VALUES (employee_id,first_name,last_name,hire_date)
SELECT * FROM employees;
```

Conditional INSERT FIRST Statements

```
--Creates a table called low_salaries with no data based on the employees table columns
CREATE TABLE low_salaries AS
SELECT employee_id, department_id, salary FROM employees WHERE 1=2;

--Creates a table called average_salaries with no data based on the employees table
columns
CREATE TABLE average_salaries AS
SELECT employee_id, department_id, salary FROM employees WHERE 1=2;

--Creates a table called high_salaries with no data based on the employees table columns
CREATE TABLE high_salaries AS
SELECT employee_id, department_id, salary FROM employees WHERE 1=2;

SELECT * FROM low_salaries;
```

```
SELECT * FROM average_salaries;
SELECT * FROM high_salaries;

INSERT FIRST
WHEN salary<5000 THEN
INTO low_salaries VALUES(employee_id, department_id, salary)
WHEN salary BETWEEN 5000 AND 10000 THEN
INTO average_salaries VALUES(employee_id, department_id, salary)
ELSE
INTO high_salaries VALUES(employee_id, department_id, salary)

SELECT * FROM employees;
```

Pivoting Insert

```
CREATE TABLE emp_sales (employee_id NUMBER(6),
                        week_id NUMBER(2),
                        sales_mon NUMBER,
                        sales_tue NUMBER,
                        sales_wed NUMBER,
                        sales_thu NUMBER,
                        sales_fri NUMBER);

CREATE TABLE emp_sales_normalized (employee_id NUMBER(6),
                                   week_id NUMBER(2),
                                   sales NUMBER,
                                   day VARCHAR2(3));

INSERT ALL
INTO emp_sales VALUES (105,23,2500,3200,4700,5600,2900)
INTO emp_sales VALUES (106,24,2740,3060,4920,5650,2800)
SELECT * FROM dual;

SELECT * FROM emp_sales;
SELECT * FROM emp_sales_normalized;

INSERT ALL
INTO emp_sales_normalized VALUES (employee_id, week_id, sales_mon, 'MON')
INTO emp_sales_normalized VALUES (employee_id, week_id, sales_tue, 'TUE')
INTO emp_sales_normalized VALUES (employee_id, week_id, sales_wed, 'WED')
INTO emp_sales_normalized VALUES (employee_id, week_id, sales_thu, 'THU')
INTO emp_sales_normalized VALUES (employee_id, week_id, sales_fri, 'FRI')
SELECT * FROM emp_sales;
```

UPDATE Statement

```
DROP TABLE employees_copy;
CREATE TABLE employees_copy AS SELECT * FROM employees;

SELECT * FROM employees_copy;
```

```
UPDATE employees_copy
SET salary = 500;

SELECT * FROM employees_copy WHERE job_id = 'IT_PROG';

UPDATE employees_copy
SET salary = 50000
WHERE job_id = 'IT_PROG';

UPDATE employees_copy
SET salary = 5, department_id = null
WHERE job_id = 'IT_PROG';

UPDATE employees_copy
SET (salary, commission_pct) = (SELECT max(salary), max(commission_pct) FROM
employees)
WHERE job_id = 'IT_PROG';

UPDATE employees_copy
SET salary = 100000
WHERE hire_date = (SELECT MAX(hire_date) FROM employees);
```

DELETE Statement

```
SELECT * FROM employees_copy;

DELETE FROM employees_copy;

DELETE employees_copy;

DELETE employees_copy
WHERE job_id = 'IT_PROG';

DELETE employees_copy
WHERE department_id IN (SELECT department_id
FROM departments
WHERE upper(department_name) LIKE'%SALES%');
```

MERGE Statement

```
SELECT * FROM employees_copy;
DELETE FROM employees_copy;
INSERT INTO employees_copy SELECT * FROM employees WHERE job_id = 'SA_REP';
UPDATE employees_copy SET first_name = 'Alex';

MERGE INTO employees_copy c
USING (SELECT * FROM employees) e
ON (c.employee_id = e.employee_id)
WHEN MATCHED THEN
```

```
        UPDATE SET
        c.first_name = e.first_name,
        c.last_name = e.last_name,
        c.department_id = e.department_id,
        c.salary = e.salary
        DELETE WHERE department_id IS NULL
WHEN NOT MATCHED THEN
        INSERT
        VALUES(e.employee_id, e.first_name, e.last_name, e.email,
        e.phone_number, e.hire_date, e.job_id, e.salary, e.commission_pct,
        e.manager_id, e.department_id);

MERGE INTO employees_copy c
USING (SELECT * FROM employees WHERE job_id = 'IT_PROG') e
ON (c.employee_id = e.employee_id)
WHEN MATCHED THEN
        UPDATE SET
        c.first_name = e.first_name,
        c.last_name = e.last_name,
        c.department_id = e.department_id,
        c.salary = e.salary
        DELETE WHERE department_id IS NULL
WHEN NOT MATCHED THEN
        INSERT
        VALUES(e.employee_id, e.first_name, e.last_name, e.email,
        e.phone_number, e.hire_date, e.job_id, e.salary, e.commission_pct,
        e.manager_id, e.department_id);

MERGE INTO employees_copy c
USING employees e
ON (c.employee_id = e.employee_id)
WHEN MATCHED THEN
        UPDATE SET
        c.first_name = e.first_name,
        c.last_name = e.last_name,
        c.department_id = e.department_id,
        c.salary = e.salary
        DELETE WHERE department_id IS NULL
WHEN NOT MATCHED THEN
        INSERT
        VALUES(e.employee_id, e.first_name, e.last_name, e.email,
        e.phone_number, e.hire_date, e.job_id, e.salary, e.commission_pct,
        e.manager_id, e.department_id);
```

(TCL) Transaction Control Language & TCL Commands

```
SELECT * FROM employees_copy;

DELETE employees_copy WHERE job_id = ‘SA_REP';
```

COMMIT and ROLLBACK Statements

```
SELECT * FROM employees_copy;
DELETE employees_copy WHERE job_id = 'SA_REP';
ROLLBACK;
UPDATE employees_copy SET first_name = 'John';
COMMIT;

UPDATE employees_copy c
        SET first_name =
        (SELECT first_name FROM employees e
        WHERE e.employee_id = c.employee_id);
INSERT INTO employees_copy
        (SELECT * FROM employees
        WHERE job_id = 'SA_REP');
CREATE TABLE temp (tmp DATE);
DROP TABLE temp;
```

Row Lock in Oracle

```
UPDATE employees_copy
SET salary = salary + 500
WHERE employee_id = 102;

SELECT employee_id,first_name,last_name,salary
FROM employees_copy
WHERE employee_id = 102;

UPDATE employees_copy
SET salary = salary + 500
WHERE employee_id = 103;

UPDATE hr.employees_copy
SET salary = salary + 1000
WHERE employee_id = 102;

UPDATE employees_copy
SET first_name = 'Alex'
WHERE employee_id = 102;
```

SAVEPOINT Statement

```
SELECT * FROM employees_copy;
SELECT * FROM jobs_copy;

--DML 1
DELETE FROM employees_copy WHERE job_id = 'IT_PROG';
SAVEPOINT A; --> Creates SavePoint A
```

```
--DML 2
UPDATE employees_copy
SET salary = 1.2 * salary;
SAVEPOINT B; --> Creates SavePoint B

--DML 3
INSERT INTO jobs_copy VALUES ('PY_DEV','Python Developer', 12000, 20000);
SAVEPOINT C; --> Creates SavePoint C

--DML 4
DELETE FROM employees_copy WHERE job_id = 'SA_REP';
SAVEPOINT D; --> Creates SavePoint D

--Rollbacks All
ROLLBACK;

--Rollbacks to SavePoint "B"
ROLLBACK TO B;

--Rollbacks to SavePoint "C"
ROLLBACK TO C;

--Rollbacks to SavePoint "A"
ROLLBACK TO SAVEPOINT A;
```

FOR UPDATE Statement

```
--CODE TO EXECUTE WITH HR
SELECT * FROM employees_copy
WHERE job_id = 'IT_PROG' FOR UPDATE;

SELECT first_name, last_name, salary
FROM employees_copy e JOIN departments d
USING(department_id)
WHERE location_id = 1400
FOR UPDATE;

SELECT first_name, last_name, salary
FROM employees_copy e JOIN departments d
USING(department_id)
WHERE location_id = 1400
FOR UPDATE OF first_name, last_name;

SELECT first_name, last_name, salary
FROM employees_copy e JOIN departments d
USING(department_id)
WHERE location_id = 1400
FOR UPDATE OF first_name, location_id NOWAIT;
```

```
SELECT first_name, last_name, salary
FROM employees_copy e JOIN departments d
USING(department_id)
WHERE location_id = 1400
FOR UPDATE OF first_name, location_id WAIT 5;

SELECT first_name, last_name, salary
FROM employees_copy e JOIN departments d
USING(department_id)
WHERE location_id = 1400
FOR UPDATE OF first_name SKIP LOCKED;
--CODE TO EXECUTE WITH SYSTEM
SELECT * FROM hr.employees_copy
WHERE job_id = 'IT_PROG';

UPDATE hr.employees_copy SET salary = 1
WHERE employee_id = 104;
UPDATE hr.employees_copy SET salary = 1
WHERE employee_id = 100;

UPDATE hr.departments SET manager_id = 100
WHERE department_id = 60;
```

FLASHBACK, PURGE

FLASHBACK Operations

```
SELECT * FROM employees_copy;

DELETE FROM employees_copy WHERE salary > 5000;

 FLASHBACK TABLE employees_copy TO TIMESTAMP sysdate - 5/(24*60);

ALTER TABLE employees_copy ENABLE ROW MOVEMENT;

SELECT dbms_flashback.get_system_change_number AS scn FROM dual;

FLASHBACK TABLE hr.employees_copy TO RESTORE POINT rp_test;

INSERT INTO employees_copy SELECT * FROM employees;

UPDATE employees_copy SET salary = 10000;

SELECT ora_rowscn, first_name FROM employees_copy;

UPDATE employees_copy
SET first_name = 'Farah'
WHERE first_name = 'Sarah';

DROP TABLE employees_copy;
```

```
--Replace your dropped object name within double quotes below.
SELECT * FROM "BIN$b3i6rakGQtC2nTeGnRs9SQ==$0";

FLASHBACK TABLE employees_copy TO BEFORE DROP;

CREATE RESTORE POINT rp_test;
FLASHBACK TABLE hr.employees_copy TO RESTORE POINT rp_test;

SELECT * FROM V$RESTORE_POINT;
```

PURGE Operations

```
SELECT * FROM recyclebin;
PURGE RECYCLEBIN;
DROP TABLE employees_copy;

SELECT * FROM employees_copy;
FLASHBACK TABLE employees_copy TO BEFORE DROP;
DROP TABLE employees_copy PURGE;

CREATE TABLE employees_copy AS SELECT * FROM employees;
CREATE TABLE employees_copy2 AS SELECT * FROM employees;
CREATE TABLE employees_copy3 AS SELECT * FROM employees;

DROP TABLE employees_copy;
DROP TABLE employees_copy2;
DROP TABLE employees_copy3;

PURGE TABLE employees_copy2;
PURGE TABLE employees_copy3;
```

Tracking Changes In Data In a Particular Time

```
SELECT * FROM employees_copy;

SELECT dbms_flashback.get_system_change_number FROM dual;

UPDATE employees_copy
SET salary = 18000
WHERE employee_id = 100;

SELECT * from employees_copy as of timestamp(sysdate - interval '2' minute) WHERE
employee_id = 100;

SELECT * from employees_copy as of scn 123123 WHERE employee_id = 100;

UPDATE employees_copy
SET salary = 15000
WHERE employee_id = 100;
```

```
SELECT versions_starttime, versions_endtime, versions_startscn, versions_endscn,
        versions_operation, versions_xid, employees_copy.*
FROM employees_copy VERSIONS BETWEEN scn MINVALUE AND MAXVALUE
WHERE employee_id = 100;

SELECT versions_starttime, versions_endtime, versions_startscn, versions_endscn,
        versions_operation, versions_xid, employees_copy.*
FROM employees_copy VERSIONS BETWEEN TIMESTAMP (sysdate - interval '5'
minute) AND sysdate
WHERE employee_id = 100;
```

NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, ON DELETE CASCADE, ON DELETE SET NULL, CHECK

NOT NULL Constraint

```
DESC jobs;
INSERT INTO jobs VALUES (100,null,1,10000);
INSERT INTO jobs VALUES (100,'My_Job',1,10000);
INSERT INTO jobs(job_id,min_salary,max_salary) VALUES (100,1,10000);
CREATE TABLE managers (manager_id NUMBER NOT NULL,
                       first_name VARCHAR2(50),
                       last_name VARCHAR2(50) CONSTRAINT
                       lname_not_null NOT NULL,
                       department_id NUMBER NOT NULL);
DESC managers;
```

UNIQUE Constraint

```
SELECT * FROM employees;
SELECT * FROM locations;
DROP TABLE managers;

CREATE TABLE managers (manager_id NUMBER CONSTRAINT mgr_mid_uk UNIQUE,
                       first_name VARCHAR2(50),
                       last_name VARCHAR2(50),
                       department_id NUMBER NOT NULL
);

INSERT INTO managers VALUES (100,'Alex','Brown',80);
INSERT INTO managers VALUES (101,'Alex','Brown',80);

CREATE TABLE managers (manager_id NUMBER CONSTRAINT mgr_mid_uk UNIQUE,
                       first_name VARCHAR2(50),
                       last_name VARCHAR2(50),
                       department_id NUMBER NOT NULL,
                       phone_number VARCHAR2(11) UNIQUE NOT NULL,
                       email VARCHAR2(100),
                       UNIQUE (email),
CONSTRAINT mgr_composite_uk UNIQUE(first_name, last_name, department_id)
);
```

```
INSERT INTO managers VALUES (100,'Alex','Brown',80,'123-456-789','abrown');
INSERT INTO managers VALUES (101,'Alex','Brown',80,'123-456-789','abrown');
INSERT INTO managers VALUES (101,'Alex','Brown',80,'123-456-780','abrown');
INSERT INTO managers VALUES (101,'Alex','Brown',80,'123-456-780','abrown2');
INSERT INTO managers VALUES (101,'Alex','Brown',90,'123-456-780','abrown2');
INSERT INTO managers VALUES (null,null,null,null,null,null);
INSERT INTO managers VALUES (null,null,null,90,null,null);
INSERT INTO managers VALUES (null,null,null,90,'123-456-781',null);
INSERT INTO managers VALUES (null,null,null,90,'123-456-782',null);
INSERT INTO managers VALUES (null,null,null,100,'123-456-782',null);

SELECT * FROM managers;
UPDATE managers SET department_id = 90 WHERE manager_id = 100;
```

PRIMARY KEY Constraint

```
DROP TABLE managers;
CREATE TABLE managers
        (manager_id NUMBER CONSTRAINT mgr_mid_uk UNIQUE,
        first_name VARCHAR2(50),
        last_name VARCHAR2(50),
        department_id NUMBER NOT NULL,
        phone_number VARCHAR2(11) UNIQUE NOT NULL,
        email VARCHAR2(100),
        UNIQUE (email),
        CONSTRAINT mgr_composite_uq UNIQUE(department_id, first_name, last_name)
);

CREATE TABLE directors
        (director_id NUMBER CONSTRAINT dir_did_pk PRIMARY KEY,
        first_name VARCHAR2(50),
        last_name VARCHAR2(50)
);

CREATE TABLE executives
        (executive_id NUMBER,
        first_name VARCHAR2(50),
        last_name VARCHAR2(50),
        CONSTRAINT dir_did_pk PRIMARY KEY (executive_id, last_name)
);

CREATE TABLE executives
        (executive_id NUMBER,
        first_name VARCHAR2(50),
        last_name VARCHAR2(50),
        CONSTRAINT exec_eid_pk PRIMARY KEY (executive_id, last_name)
);

INSERT INTO directors VALUES(100,'John','Goodman');
INSERT INTO executives VALUES(100,'John',null);
```

```
DROP TABLE executives;
DROP TABLE directors;

FOREIGN KEY Constraint
DROP TABLE managers;
CREATE TABLE managers
        (manager_id NUMBER CONSTRAINT mgr_mid_uk UNIQUE,
        first_name VARCHAR2(50),
        last_name VARCHAR2(50),
        department_id NUMBER NOT NULL,
        phone_number VARCHAR2(11) UNIQUE NOT NULL,
        email VARCHAR2(100),
        UNIQUE(email),
        CONSTRAINT mgr_composite_uq UNIQUE(department_id, first_name, last_name)
);

SELECT * FROM employees;
SELECT * FROM employees_copy;

CREATE TABLE managers
        (manager_id NUMBER CONSTRAINT mgr_mid_pk PRIMARY KEY,
        first_name VARCHAR2(50),
        last_name VARCHAR2(50),
        department_id NUMBER NOT NULL,
        phone_number VARCHAR2(11) UNIQUE NOT NULL,
        email VARCHAR2(100),
        UNIQUE (email),
        CONSTRAINT mgr_emp_fk FOREIGN KEY (manager_id) REFERENCES
employees_copy (employee_id)
);
        DROP TABLE employees_copy;
CREATE TABLE employees_copy
        (employee_id NUMBER(6) CONSTRAINT emp_cpy_eid_pk PRIMARY KEY,
        first_name VARCHAR2(20),
        last_name VARCHAR2(20),
        department_id NUMBER(4)
);

INSERT INTO employees_copy
SELECT employee_id, first_name, last_name, department_id
FROM employees;

SELECT * FROM employees_copy;
INSERT INTO managers VALUES (80, 'John', 'King', 90, '123-456-789','jking');
INSERT INTO managers VALUES (100, 'John', 'King', 90, '123-456-789','jking');

UPDATE managers
SET manager_id = 70
WHERE manager_id = 100;

DELETE FROM employees_copy
WHERE employee_id = 100;
```

```
CREATE TABLE managers
        (manager_id NUMBER CONSTRAINT mgr_mid_uq UNIQUE,
        first_name VARCHAR2(50),
        last_name VARCHAR2(50),
        department_id NUMBER NOT NULL,
        phone_number VARCHAR2(11) UNIQUE NOT NULL,
        email VARCHAR2(100),
        UNIQUE (email),
        CONSTRAINT mgr_emp_fk FOREIGN KEY (manager_id) REFERENCES
employees_copy (employee_id));

INSERT INTO managers values (103, 'John', 'King', 90, '123-456-789','jking');

CREATE TABLE managers
        (manager_id NUMBER CONSTRAINT mgr_mid_uq UNIQUE,
        first_name VARCHAR2(50),
        last_name VARCHAR2(50),
        department_id NUMBER NOT NULL,
        phone_number VARCHAR2(11) UNIQUE NOT NULL,
        email VARCHAR2(100),
        UNIQUE (email),
        CONSTRAINT mgr_emp_fk FOREIGN KEY (manager_id) REFERENCES
employees_copy (employee_id),
        CONSTRAINT mgr_names_fk FOREIGN KEY (first_name, last_name)
REFERENCES employees_copy(first_name, last_name)
);

CREATE TABLE employees_copy (employee_id NUMBER(6) CONSTRAINT
emp_cpy_eid_pk PRIMARY KEY,
        first_name VARCHAR(20),
        last_name VARCHAR(20),
        department_id NUMBER(4),
        CONSTRAINT emp_cpy_names_uk UNIQUE (first_name, last_name));
```

The ON DELETE CASCADE ON DELETE SET NULL Clause

```
DROP TABLE managers;
CREATE TABLE managers
        (manager_id NUMBER CONSTRAINT mgr_mid_pk PRIMARY KEY,
        first_name VARCHAR2(50),
        last_name VARCHAR2(50),
        department_id NUMBER NOT NULL,
        phone_number VARCHAR2(11) UNIQUE NOT NULL,
        email VARCHAR2(100),
        UNIQUE(email),
        CONSTRAINT mgr_emp_fk FOREIGN KEY (manager_id) REFERENCES
employees_copy (employee_id)
);

DELETE FROM managers;
```

```
INSERT INTO managers values (103, 'John', 'King', 90, '123-456-789','jking');
INSERT INTO managers values (104, 'John2', 'King', 90, '123-456-780','jking2');
INSERT INTO managers values (105, 'John3', 'King', 90, '123-456-781','jking3');

SELECT * FROM employees_copy;
SELECT * FROM managers;

CREATE TABLE managers
        (manager_id NUMBER,
        first_name VARCHAR2(50),
        last_name VARCHAR2(50),
        department_id NUMBER NOT NULL,
        phone_number VARCHAR2(11) UNIQUE NOT NULL,
        email VARCHAR2(100),
        UNIQUE (email),
        CONSTRAINT mgr_emp_fk FOREIGN KEY (manager_id) REFERENCES
employees_copy (employee_id) ON DELETE SET NULL
);

DELETE FROM employees_copy
WHERE employee_id = 103;

DELETE FROM employees_copy
WHERE employee_id = 150;

CREATE TABLE managers
        (manager_id NUMBER,
        first_name VARCHAR2(50),
        last_name VARCHAR2(50),
        department_id NUMBER NOT NULL,
        phone_number VARCHAR2(11) UNIQUE NOT NULL,
        email VARCHAR2(100),
        UNIQUE (email),
        CONSTRAINT mgr_emp_fk FOREIGN KEY (manager_id) REFERENCES
employees_copy (employee_id) ON DELETE CASCADE
);

DELETE FROM employees_copy
WHERE employee_id = 104;

UPDATE employees_copy
SET employee_id = 300
WHERE employee_id = 105;
```

CHECK Constraint (Code Samples)

```
CREATE TABLE managers2
        (manager_id NUMBER,
        first_name VARCHAR2(50),
        salary NUMBER,
        CONSTRAINT salary_check CHECK (salary > 100 AND salary < 50000)
);
```

```
INSERT INTO managers2 VALUES(1, 'Steven', 50);
INSERT INTO managers2 VALUES(1, 'Steven', 500);
UPDATE managers2
SET salary = 20
WHERE manager_id = 1;

DROP TABLE managers2;

CREATE TABLE managers2 (
        manager_id NUMBER,
        first_name VARCHAR2(50),
        salary NUMBER,
        email VARCHAR2(100),
        CONSTRAINT demo_check CHECK (salary > 100 AND salary < 50000 AND
upper(email) LIKE '%.COM')
);

INSERT INTO managers2 VALUES (1, 'Steven', 500,'thisisademoemail.xyz);
INSERT INTO managers2 VALUES (1, 'Steven', 500,'thisisademoemail.com');
```

Adding Constraints via ALTER TABLE Statements

```
DROP TABLE managers;
DROP TABLE employees_copy;

CREATE TABLE employees_copy AS SELECT * FROM employees;

ALTER TABLE employees_copy ADD CONSTRAINT emp_cpy_email_uk UNIQUE (email);

ALTER TABLE employees_copy ADD CONSTRAINT emp_cpy_names_uk UNIQUE
(first_name, last_name);

ALTER TABLE employees_copy ADD UNIQUE (phone_number);
ALTER TABLE employees_copy ADD CHECK (salary > 10000);
ALTER TABLE employees_copy ADD CHECK (salary > 1000);

ALTER TABLE employees_copy ADD CONSTRAINT emp_cpy_emp_id_pk PRIMARY
KEY (employee_id);

ALTER TABLE employees_copy ADD CONSTRAINT emp_cpy_dept_fk FOREIGN KEY
(department_id) REFERENCES departments(department_id);

ALTER TABLE employees_copy MODIFY salary CONSTRAINT emp_cpy_salary_nn NOT
NULL;

ALTER TABLE employees_copy MODIFY last_name NOT NULL;
ALTER TABLE employees_copy MODIFY first_name NOT NULL;
```

Dropping (Removing) Constraints

```
SELECT * FROM employees_copy;

CREATE TABLE managers
        (manager_id NUMBER CONSTRAINT mgr_mid_pk PRIMARY KEY,
        first_name VARCHAR2(50),
        last_name VARCHAR2(50),
        department_id NUMBER NOT NULL,
        phone_number VARCHAR2(11) UNIQUE NOT NULL,
        email VARCHAR2(100),
        UNIQUE(email),
        CONSTRAINT mgr_emp_fk FOREIGN KEY (manager_id) REFERENCES
employees_copy (employee_id)
);

DROP TABLE managers;

ALTER TABLE employees_copy DROP CONSTRAINT emp_cpy_emp_id_pk;
ALTER TABLE employees_copy DROP CONSTRAINT emp_cpy_emp_id_pk CASCADE;
ALTER TABLE employees_copy DROP PRIMARY KEY CASCADE;
ALTER TABLE employees_copy DROP CONSTRAINT SYS_C008689 ONLINE;
```

Cascading Constraints

```
DROP TABLE employees_copy;
DROP TABLE departments_copy;
CREATE TABLE employees_copy AS SELECT * FROM employees;
CREATE TABLE departments_copy AS SELECT * FROM departments;

ALTER TABLE departments_copy ADD CONSTRAINT dept_id_pk PRIMARY KEY
(department_id);

ALTER TABLE departments_copy ADD CONSTRAINT dept_cpy_id_pk PRIMARY KEY
(department_id);

ALTER TABLE employees_copy
ADD CONSTRAINT emp_dept_cpy_fk FOREIGN KEY (department_id) REFERENCES
departments_copy (department_id);

ALTER TABLE departments_copy DROP COLUMN department_id;
ALTER TABLE departments_copy DROP COLUMN department_id CASCADE
CONSTRAINTS;

ALTER TABLE employees_copy ADD UNIQUE (first_name, last_name);
ALTER TABLE employees_copy DROP COLUMN last_name;
ALTER TABLE employees_copy DROP COLUMN last_name CASCADE CONSTRAINTS;
```

Renaming Constraints

```
CREATE TABLE employees_copy AS SELECT * FROM employees;

ALTER TABLE employees_copy RENAME CONSTRAINT SYS_C008743 TO email_no;
```

Disabling Constraints

```
DROP TABLE employees_copy;
DROP TABLE departments_copy;
CREATE TABLE departments_copy AS SELECT * FROM departments;
CREATE TABLE employees_copy AS SELECT * FROM employees;

ALTER TABLE departments_copy
ADD CONSTRAINT dept_cpy_id_pk PRIMARY KEY(department_id);

ALTER TABLE employees_copy
ADD CONSTRAINT emp_dept_copy_fk FOREIGN KEY(department_id) REFERENCES
departments_copy (department_id);

UPDATE departments_copy
SET department_name = null
WHERE department_id = 10;

ALTER TABLE departments_copy
DISABLE CONSTRAINT SYS_C008762;

UPDATE departments_copy
SET department_id = 5
WHERE department_id = 80;

ALTER TABLE departments_copy
DISABLE CONSTRAINT dept_cpy_id_pk;

ALTER TABLE departments_copy
DISABLE CONSTRAINT dept_cpy_id_pk CASCADE;

ALTER TABLE departments_copy
ADD CONSTRAINT dept_cpy_id_pk PRIMARY KEY (department_id) DISABLE;
```

Enabling Constraints

```
INSERT INTO departments_copy VALUES (10,'TempDept',100,1700);

ALTER TABLE departments_copy ENABLE CONSTRAINT dept_cpy_id_pk;

SELECT * FROM departments_copy ORDER BY department_id;

DELETE FROM departments_copy WHERE department_name = 'TempDept';
```

Status of Constraints

```
DROP TABLE departments_copy;
CREATE TABLE departments_copy AS SELECT * FROM departments;
ALTER TABLE departments_copy ADD CONSTRAINT dept_cpy_id_pk PRIMARY KEY
(department_id);

ALTER TABLE departments_copy DISABLE CONSTRAINT dept_cpy_id_pk;
ALTER TABLE departments_copy DISABLE NOVALIDATE CONSTRAINT
dept_cpy_id_pk;

ALTER TABLE departments_copy DISABLE VALIDATE CONSTRAINT dept_cpy_id_pk;
ALTER TABLE departments_copy ENABLE CONSTRAINT dept_cpy_id_pk;
ALTER TABLE departments_copy ENABLE NOVALIDATE CONSTRAINT dept_cpy_id_pk;

UPDATE departments_copy SET department_id = 10 WHERE department_id = 20;

SELECT * FROM departments_copy;

DELETE FROM departments_copy WHERE department_id = 10;
UPDATE departments_copy SET department_name ='Temp' WHERE department_id = 30;
UPDATE departments_copy SET department_id = NULL WHERE department_id = 40;
UPDATE departments_copy SET department_id = NULL WHERE department_id = 50;
```

Deferring Constraints

```
DROP TABLE departments_copy;
CREATE TABLE departments_copy AS SELECT * FROM departments;

ALTER TABLE departments_copy
ADD CONSTRAINT dept_cpy_id_pk PRIMARY KEY (department_id) DEFERRABLE
INITIALLY DEFERRED;

INSERT INTO departments_copy VALUES (10, 'Temp Department', 200, 1700);

SET CONSTRAINT dept_cpy_id_pk IMMEDIATE;
SET CONSTRAINT dept_cpy_id_pk DEFERRED;
SET CONSTRAINTS ALL IMMEDIATE;
ALTER SESSION SET CONSTRAINTS = IMMEDIATE;

ALTER TABLE departments_copy DROP CONSTRAINT dept_cpy_id_pk;
ALTER TABLE departments_copy

ADD CONSTRAINT dept_cpy_id_pk PRIMARY KEY (department_id) NOT DEFERRABLE;
```

VİEWS

Creating Simple Views

```
SELECT * FROM employees WHERE department_id = 90;

CREATE VIEW empvw90 AS
SELECT * FROM employees WHERE department_id = 90;

SELECT * FROM empvw90;
SELECT * FROM empvw90 WHERE salary < 20000;

CREATE VIEW empvw20 AS
SELECT employee_id, first_name, last_name FROM employees WHERE department_id =
20;

SELECT * FROM empvw20;
SELECT first_name, last_name FROM empvw20;

CREATE VIEW empvw30 AS
SELECT employee_id e_id, first_name name, last_name surname
FROM employees WHERE department_id = 30;
SELECT * FROM empvw30;

CREATE VIEW empvw40 (e_id, name, surname, email) AS
SELECT employee_id, first_name, last_name, email
FROM employees WHERE department_id = 40;
SELECT * FROM empvw40;

CREATE VIEW empvw41 (e_id, name, surname, email) AS
SELECT employee_id eid, first_name, last_name, email
FROM employees WHERE department_id = 40;
SELECT * FROM empvw41;
```

Creating Complex Views

```
CREATE VIEW emp_cx_vw (DNAME, MIN_SAL, MAX_SAL) AS
SELECT distinct upper(department_name), min(salary), max(salary)
FROM employees e JOIN departments d
USING(department_id)
GROUP BY department_name;

SELECT * FROM emp_cx_vw;
```

Modifying Views

```
SELECT * FROM empvw30;

CREATE OR REPLACE VIEW empvw30 AS
SELECT employee_id e_id, first_name name, last_name surname, job_id
FROM employees WHERE department_id = 30;

CREATE OR REPLACE VIEW empvw30 AS
SELECT employee_id e_id, first_name||' '||last_name name, job_id
FROM employees WHERE department_id = 30;
```

Performing DML Operations with Views

```
DROP TABLE employees_copy CASCADE CONSTRAINTS;
CREATE TABLE employees_copy AS SELECT * FROM employees;

CREATE VIEW empvw60 AS
SELECT employee_id, first_name, last_name, email, hire_date, job_id
FROM employees_copy
WHERE department_id = 60;

SELECT * FROM employees_copy;
SELECT * FROM employees_copy WHERE department_id = 60;
SELECT * FROM empvw60;

INSERT INTO empvw60 VALUES (213,'Alex','Hummel','AHUMMEL',sysdate,'IT_PROG');

CREATE OR REPLACE VIEW empvw60 AS
SELECT employee_id, first_name, last_name, email, hire_date, job_id, department_id
FROM employees_copy
WHERE department_id = 60;

INSERT INTO empvw60 VALUES
(214,'Alex','Hummel','AHUMMEL',sysdate,'IT_PROG',60);
UPDATE empvw60 SET job_id = 'SA_MAN' where employee_id = 214;
DELETE FROM empvw60;

CREATE OR REPLACE VIEW empvw60 AS
SELECT distinct employee_id, first_name, last_name, email, hire_date, job_id,
department_id
FROM employees_copy
WHERE department_id = 60;

CREATE OR REPLACE VIEW empvw60 AS
SELECT rownum rn, employee_id, first_name, last_name, email, hire_date, job_id,
department_id
FROM employees_copy
WHERE department_id = 60;
```

```
INSERT INTO empvw60 VALUES
(1,214,'Alex','Hummel','AHUMMEL',sysdate,'IT_PROG',60);

CREATE OR REPLACE VIEW empvw60 AS
SELECT employee_id, first_name, last_name, email, hire_date,
job_id, department_id, salary*12 annual_salary
FROM employees_copy
WHERE department_id = 60;

INSERT INTO empvw60 VALUES
(214,'Alex','Hummel','AHUMMEL',sysdate,'IT_PROG',60, 120000);
UPDATE empvw60 SET job_id = 'SA_MAN' where employee_id = 107;
DELETE empvw60 WHERE employee_id = 107;
```

Using the WITH CHECK OPTION Clause in SQL

```
DROP TABLE employees_copy;
CREATE TABLE employees_copy AS SELECT * FROM employees;

CREATE OR REPLACE VIEW empvw80 AS
SELECT employee_id, first_name, last_name, email, hire_date, job_id
FROM employees_copy
WHERE department_id = 80;

SELECT * FROM empvw80;

INSERT INTO empvw80 VALUES (215,'John','Brown','JBROWN',sysdate,'SA_MAN');

SELECT * FROM employees_copy;

CREATE OR REPLACE VIEW empvw80 AS
SELECT employee_id, first_name, last_name, email, hire_date, job_id
FROM employees_copy
WHERE department_id = 80
WITH CHECK OPTION CONSTRAINT emp_dept80_chk;

INSERT INTO empvw80 VALUES (216,'John2','Brown2','JBROWN2',sysdate,'SA_MAN');

CREATE OR REPLACE VIEW empvw80 AS
SELECT employee_id, first_name, last_name, email, hire_date, job_id, department_id
FROM employees_copy
WHERE department_id = 80
WITH CHECK OPTION;

INSERT INTO empvw80 VALUES (217,'John3','Brown3','JBROWN3',sysdate,'SA_MAN',
80);
INSERT INTO empvw80 VALUES (218,'John4','Brown4','JBROWN4',sysdate,'SA_MAN',
60);
```

```
CREATE OR REPLACE VIEW empvw80 AS
SELECT employee_id, first_name, last_name, email, hire_date, job_id, department_id
FROM employees_copy
WHERE department_id = 80
AND job_id = 'SA_MAN'
WITH CHECK OPTION;

INSERT INTO empvw80 VALUES (219,'John3','Brown3','JBROWN3',sysdate,'IT_PROG',
80);
UPDATE empvw80 SET first_name = 'Steve' WHERE employee_id = 217;
UPDATE empvw80 SET department_id = 70 WHERE employee_id = 217;

SELECT * FROM user_constraints WHERE table_name = ‘EMPVW80';
```

Using the WITH READ ONLY Clause on Views

```
CREATE OR REPLACE VIEW empvw80 AS
SELECT employee_id, first_name, last_name, email, hire_date, job_id, department_id
FROM employees_copy
WHERE department_id = 80
AND job_id = 'SA_MAN'
WITH READ ONLY;

SELECT * FROM empvw80;

INSERT INTO empvw80 VALUES (219,'John3','Brown3','JBROWN3',sysdate,'IT_PROG',
80);
UPDATE empvw80 SET first_name = 'Steve' WHERE employee_id = 217;
DELETE FROM empvw80 WHERE employee_id = 217;
```

Dropping Views

```
DROP VIEW empvw20;
DROP VIEW empvw30;
DROP VIEW empvw40;
DROP VIEW empvw41;
DROP VIEW empvw60;

SELECT * FROM user_constraints WHERE table_name = 'EMPVW80';

DROP VIEW empvw80;
```

SEQUENCES

Creating Sequences

```
CREATE SEQUENCE employee_seq
START WITH 100
INCREMENT BY 3
MAXVALUE 50
CACHE 30
NOCYCLE;

CREATE SEQUENCE employee_seq
START WITH 100
INCREMENT BY 3
MAXVALUE 99999
CACHE 30
NOCYCLE;
```

Modifying Sequences

```
CREATE SEQUENCE employee_seq
START WITH 100
INCREMENT BY 2
MAXVALUE 99999
CACHE 30
NOCYCLE;

ALTER SEQUENCE employee_seq
START WITH 100
INCREMENT BY 5
MAXVALUE 99999
CACHE 30
NOCYCLE;


ALTER SEQUENCE employee_seq
INCREMENT BY 5
MAXVALUE 99999
CACHE 30
NOCYCLE;

ALTER SEQUENCE employee_seq
INCREMENT BY 4
NOCYCLE;
```

Using Sequences

```
DROP SEQUENCE employee_seq;

CREATE SEQUENCE employee_seq
START WITH 100
INCREMENT BY 3
MAXVALUE 99999
CACHE 30
NOCYCLE;

SELECT employee_seq.CURRVAL FROM dual;
SELECT employee_seq.NEXTVAL FROM dual;

INSERT INTO employees (employee_id, last_name, email, hire_date, job_id)
VALUES (employee_seq.NEXTVAL, 'Smith','SMITH5',sysdate,'IT_PROG');

SELECT * FROM employees;

SELECT employee_seq.NEXTVAL FROM employees;
```

Using Sequences as a Default Value

```
CREATE TABLE temp (e_id INTEGER DEFAULT employee_seq.NEXTVAL, first_name
VARCHAR2(50));

INSERT INTO temp(first_name) VALUES ('Alex');
SELECT * FROM temp;

DROP TABLE temp;

CREATE TABLE temp (e_id INTEGER DEFAULT employee_seq.CURRVAL, first_name
VARCHAR2(50));

SELECT employee_seq.NEXTVAL FROM dual;
INSERT INTO temp(e_id,first_name) VALUES(NULL,'Alex');
```