ORACLE SQL 11G/12C

Version: 1.0

Contents

1. SQL Overview

1.1. What is SQL?

SQL is Structured Query Language, which is a computer language for storing, manipulating and retrieving data stored in relational database.

SQL is the standard language for Relation Database System. All relational database management systems like MySQL, MS Access, Oracle, Sybase, Informix, postgres and SQL Server use SQL as standard database language.

Also, they are using different dialects, such as:

- MS SQL Server using T-SQL,
- Oracle using PL/SQL,
- MS Access version of SQL is called JET SQL (native format) etc.

1.2 Why SQL?

- Allows users to access data in relational database management systems.
- Allows users to describe the data.
- Allows users to define the data in database and manipulate that data.
- Allows to embed within other languages using SQL modules, libraries & pre-compilers.
- Allows users to create and drop databases and tables.
- Allows users to create view, stored procedure, functions in a database.
- Allows users to set permissions on tables, procedures and views

1.3 History.

- 1970 -- Dr. E. F. "Codd" of IBM is known as the father of relational databases. He described a relational model for databases.
- 1974 -- Structured Query Language appeared.
- 1978 -- IBM worked to develop Codd's ideas and released a product named System/R.
- 1986 -- IBM developed the first prototype of relational database and standardized by ANSI. The first relational database was released by Relational Software and its later becoming Oracle.

1.4 SQL Commands.

The standard SQL commands to interact with relational databases are CREATE, SELECT, INSERT, UPDATE, DELETE and DROP. These commands can be classified into groups based on their nature:

1.4.1 DDL - Data Definition Language:

COMMAND	DESCRIPTION
CREATE	Creates a new table, a view of a table, or other object in database
ALTER	Modifies an existing database object, such as a table.
DROP	Deletes an entire table, a view of a table or other object in the database.
TRUNCATE	Deletes an entire ROW OF table OR view of a table .

1.4.2 DML - Data Manipulation Language.

COMMAND	DESCRIPTION
INSERT	Creates a record.(NEW ROW)
UPDATE	Modifies records OR ROW.
DELETE	Deletes records or row.
MERGE	Merge two table on certain condition.

1.4.3 DCL -Data Control Language.

COMMAND	DESCRIPTION
GRANT	Gives a privilege to user
REVOKE	Takes back privileges granted from user

1.4.4 TCL - Transaction Control Language.

COMMAND	DESCRIPTIO
COMMIT	For making the changes permanent in db.
ROLLBACK	For reversing the changes that you made
SAVEPOINT	For partially reversing dml operation.

1.4.5 DQL-Data Query Language

Ccommand	DESCRIPTION
SELECT	Retrieves certain records from one or more tables

1.5 What is NULL value?

A NULL value in a table is a value in a field that appears to be blank, which means a field with a NULL value is a field with no value.

It is very important to understand that a NULL value is different than a zero value or a field that contains spaces. A field with a NULL value is one that has been left blank during record creation.

Any Arithmatic operation with NULL values will always return a null value.

2. SQL RDBMS Concepts.

2.1 What is RDBMS?

RDBMS stands for Relational Database Management System. RDBMS is the basis for SQL and for all modern database systems like MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access. A Relational database management system (RDBMS) is a database management system (DBMS) that is based on the relational model as introduced by E. F. Codd.

2.2 What is table?

The data0 in RDBMS is stored in database objects called tables. The table is a collection of related data entries and it consists of columns and rows. Remember, a table is the most common and simplest form of data storage in a relational database.

2.3 What is column?

A column is a vertical entity in a table that contains all information associated with a specific field in a table.

2.4 What is field?

Every table is broken up into smaller entities called fields. The fields in the CUSTOMERS table consist of ID, NAME, AGE, ADDRESS and SALARY.

A field is a column in a table that is designed to maintain specific information about every record in the table.

2.5 What is record or row?

A record, also called a row of data, is each individual entry that exists in a table.Following is a single row of data or record in the REGIONS table:

```
SQL> SELECT * FROM REGIONS;

 REGION_ID REGION_NAME
---------- -------------------------
         1 Europe
         2 Americas
         3 Asia
         4 Middle East and Africa
```

A record is a horizontal entity in a table.

2.6 What is pseudo column?

A **pseudocolumn** behaves like a table column, but is not actually stored in the table. You can select from pseudocolumns, but you cannot insert, update, or delete their values.

2.6.1 ROW ID PSEUDO COLUMN

For each row in the database, the **ROWID** pseudocolumn returns the address of the row. Oracle Database rowid values contain information necessary to locate a row.

Rowid values have several important uses:

- They are the fastest way to access a single row.
- They can show you how the rows in a table are stored.
- They are unique identifiers for rows in a table.
- You should not use **ROWID** as the primary key of a table. If you delete and reinsert a row with the Import and Export utilities, for example, then its rowid may change. If you delete a row, then Oracle may reassign its rowid to a new row inserted later.
- Although you can use the **ROWID** pseudocolumn in the **SELECT** and **WHERE** clause of a query, these pseudocolumn values are not actually stored in the database. You cannot insert, update, or delete a value of the **ROWID** pseudocolumn.

Example:

```
SQL> select rowid,employee_id from employees
  2  where department_id=20;

ROWID              EMPLOYEE_ID
------------------ -----------
AACHH8AAZAAAB0bAAD         201
AACHH8AAZAAAB0bAAE         202
```

2.6.2 ROWNUM PSEUDO COLUMN

For each row returned by a query, the **ROWNUM** pseudocolumn returns a number indicating the order in which Oracle selects the row from a table or set of joined rows. The first row selected has a **ROWNUM** of 1, the second has 2, and so on.

You can use **ROWNUM** to limit the number of rows returned by a query, as in this example:

```
SQL> select employee_id,last_name,rownum from employees where rownum<4;

EMPLOYEE_ID LAST_NAME                 ROWNUM
----------- ------------------------- ----------
        198 OConnell                           1
        199 Grant                              2
        200 Whalen                             3
```

If you embed the **ORDER BY** clause in a subquery and place the **ROWNUM** condition in the top-level query, then you can force the **ROWNUM** condition to be applied after the ordering of the rows. For example, the following query returns the employees with the 5 smallest employee numbers. This is sometimes referred to as **top-N reporting**.

```
SQL> SELECT * FROM
  2     (SELECT employee_id,salary FROM employees ORDER BY salary desc)
  3     WHERE ROWNUM <= 5;

EMPLOYEE_ID     SALARY
----------- ----------
        100      26400
        101      17000
        102      17000
        145      14000
        146      13500
```

3. Retrieving Data Using SELECT Statement

3.1 DESCRIBE.

3.1.1 How to view the structure of the table ??

Describe employee;

OR

Desc employees;

The Above both command are equal and give the same output i.e structure of a table along with data types.

```
SQL> desc employees;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 EMPLOYEE_ID                               NOT NULL NUMBER(6)
 FIRST_NAME                                         VARCHAR2(20)
 LAST_NAME                                 NOT NULL VARCHAR2(25)
 EMAIL                                     NOT NULL VARCHAR2(25)
 PHONE_NUMBER                                       VARCHAR2(20)
 HIRE_DATE                                 NOT NULL DATE
 JOB_ID                                    NOT NULL VARCHAR2(10)
 SALARY                                             NUMBER(8,2)
 COMMISSION_PCT                                     NUMBER(2,2)
 MANAGER_ID                                         NUMBER(6)
 DEPARTMENT_ID                                      NUMBER(4)
```

3.2 SELECT STATEMENT.

3.2.1 How to retrive all the columns from table.???

For retrieving all the columns from the table we use an special symbol of ‘ * ’ in sql . For example if we want select all details of employee 100.

```
SQL> select * from employees where employee_id=100;

EMPLOYEE_ID FIRST_NAME           LAST_NAME
----------- -------------------- -------------------------
EMAIL                     PHONE_NUMBER         HIRE_DATE JOB_ID         SALARY
------------------------- -------------------- --------- ---------- ----------
COMMISSION_PCT MANAGER_ID DEPARTMENT_ID
-------------- ---------- -------------
        100 Steven               King
SKING                     515.123.4567         17-JUN-87 AD_PRES         26400
                                    90
```

How to retrive selected columns from table?

You can select any column just by mentioning the column name.i.e

```
SQL> select employee_id,salary from employees;

EMPLOYEE_ID     SALARY
----------- ----------
        198       2600
        199       2600
        200       4400
        201      13000
        202       6000
        203       6500
```

3.2.2 How to use arithmetic operators in SQL statements?

Here in this example we are increasing salary by 300.

```
SQL> select salary,salary+300 from employees;

     SALARY SALARY+300
---------- ----------
       2600       2900
       2600       2900
       4400       4700
      13000      13300
       6000       6300
       6500       6800
```

NOTE: Always remember any operation with null value will give null.

For example

Select employee_id,commission_pct,commission_pct+3 from employees;

3.2.3 Operator Precedence in SQL

In Oracle SQL operators will always follow **BODMAS** Rule
(Bracket,pOwer,Division,Multiplication,Addition,Subtraction).

Consider the following example.Both give different answer.

```
SQL> select last_name,12*salary+100 from employees;

LAST_NAME                 12*SALARY+100
------------------------- -------------
OConnell                          31300
Grant                             31300
Whalen                            52900
Hartstein                        156100
Fay                               72100
```

```
SQL> select last_name,12*(salary+100) from employees;

LAST_NAME                 12*(SALARY+100)
------------------------- ---------------
OConnell                            32400
Grant                               32400
Whalen                              54000
Hartstein                          157200
Fay                                 73200
Mavris                              79200
```

3.2.4 How to define a column Alias.??

SELECT last_name AS name, commission_pct comm FROM employees;

```
SQL>  SELECT last_name AS name, commission_pct  comm FROM   employees;

NAME                           COMM
-------------------------- ----------
OConnell
Grant
Whalen
Hartstein
Fay
Mavris
```

or

SELECT last_name "Name" , salary*12 "Annual Salary" FROM employees;

```
SQL> select last_name "NAME",salary*12 "ANNUAL SALARY" from employees;

NAME                      ANNUAL SALARY
------------------------- -------------
OConnell                          31200
Grant                             31200
Whalen                            52800
Hartstein                        156000
Fay                               72000
Mavris                            78000
```

3.3 Concatenation Operator

It is used to concat some string with the column field in the select statement for the desired output.for example.

```
SQL> select last_name||' having salary of amount '||salary as "EMP_DET" from employees;

EMP_DET
-----------------------------------------------------------------------------------------
OConnell having salary of amount 2600
Grant having salary of amount 2600
Whalen having salary of amount 4400
Hartstein having salary of amount 13000
Fay having salary of amount 6000
Mavris having salary of amount 6500
```

3.4 DISTINCT.

How to suppress a duplicate rows of column???just consider following example.

```
DEPARTMENT_ID
-------------
           50
           50
           10
           20
           20
           40
           70
          110
          110
           90
           90
```

```
SQL> select distinct department_id from employees;

DEPARTMENT_ID
-------------
          100
           30

           20
           70
           90
          110
           50
           40
           80
           10
```

4. Restricting And Sorting Data.

1- Using the WHERE Clause to restrict the data.

```
SQL> select last_name,job_id,department_id from employees where department_id=20;

LAST_NAME                 JOB_ID     DEPARTMENT_ID
------------------------- ---------- -------------
Hartstein                 MK_MAN                20
Fay                       MK_REP                20
```

2. where clause with character string.

```
SQL> select last_name,job_id,department_id from employees where last_name='Whalen';

LAST_NAME                 JOB_ID     DEPARTMENT_ID
------------------------- ---------- -------------
Whalen                    AD_ASST               10
```

All character searches are case sensitive. And Date is Format sensitive. In the following example, no rows are returned because the EMPLOYEES table stores all the last names in mixed case.

```
SQL> select last_name,job_id,department_id from employees where last_name='WHALEN';

no rows selected
```

3. Comparison Operator.

= Equal to

> Greater than

>= Greater than or equal to

< Less than

<= Less than or equal to

<> Not equal to

Between---And

They are used in where clause as follows.

```
SQL> select last_name ,salary from employees where salary<5000;

LAST_NAME                      SALARY
------------------------- ----------
OConnell                         2600
Grant                            2600
Whalen                           4400
Austin                           4800
Pataballa                        4800
Lorentz                          4203
Khoo                             3100
```

4. **BETWEEN AND--to** find out the range between 2 values.

```
SQL> select last_name,salary from employees where salary between 3000 and 5000;

LAST_NAME                       SALARY
------------------------- ----------
Whalen                           4400
Austin                           4800
Pataballa                        4800
Lorentz                          4203
Khoo                             3100
Nayer                            3200
```

The above query and below query both will give the same output.

- **NOTE: when using BETWEEN –AND always mention the lower value first.**

```
SQL> select last_name,salary from employees
  2  where salary<=5000 and salary>=3000;

LAST_NAME                      SALARY
------------------------- ----------
Whalen                           4400
Austin                           4800
Pataballa                        4800
Lorentz                          4203
Khoo                             3100
Nayer                            3200
```

5. **IN** condition--match any list of the values

```
SQL> select employee_id,last_name from employees where employee_id in(100,102);

EMPLOYEE_ID LAST_NAME
----------- -------------------------
        102 De Haan
        100 King
```

The above query we can run with ' or ' also performance of both query is same.

```
SQL> select employee_id,last_name from employees where employee_id=100 or employee_id=102;

EMPLOYEE_ID LAST_NAME
----------- -------------------------
        102 De Haan
        100 King
```

6. LIKE condition--Match a character pattern (wild card searches)

- ' % ' denotes zero or many characters.
- '_' denotes one character.

Example .If you wanna know details of those employees whose last_name ends with character 'n'.

```
SQL> select last_name from employees where last_name like '%n';

LAST_NAME
-------------------------
Atkinson
Austin
Bernstein
Chen
De Haan
Doran
```

Example 2.If you wanna know employee last_name whose second character is 'r' and last character is 'e'.

```
SQL> select last_name from employees where last_name like '_r%e';

LAST_NAME
-------------------------
Greene
```

You can use the ESCAPE identifier to search for actual % and _ symbols.

When you need to have an exact match for the actual % and _ characters, use the ESCAPE option. This option specifies what the escape character is. If you want to search for strings that contain SA_.
you can use the following SQL statement.

```
SQL> select employee_id,last_name,job_id from employees where job_id like '%SA\_%' ESCAPE '\';

EMPLOYEE_ID LAST_NAME                 JOB_ID
----------- ------------------------- ----------
        145 Russell                   SA_MAN
        146 Partners                  SA_MAN
        147 Errazuriz                 SA_MAN
        148 Cambrault                 SA_MAN
        149 Zlotkey                   SA_MAN
```

The ESCAPE option identifies the backslash (\) as the escape character. In the pattern, the escape character precedes the underscore (_). This causes the OracleServer to interpret the underscore literally.

7. NULL VALUES.(Columns which have null value how to select them in where clause).

```
SQL> select employee_id ,manager_id from employees where manager_id is null;

EMPLOYEE_ID MANAGER_ID
----------- ----------
        100
```

```
SQL> select employee_id,commission_pct from employees where commission_pct is not null;

EMPLOYEE_ID COMMISSION_PCT
----------- --------------
        145             .4
        146             .3
        147             .3
        148             .3
        149             .2
```

8. Logical Conditions

1. AND
2. OR
3. NOT

AND--returns true if both conditions are ture.

```
SQL> select last_name,job_id,salary from employees where salary>10000 and job_id like '%MAN%';

LAST_NAME                 JOB_ID         SALARY
------------------------- ---------- ----------
Hartstein                 MK_MAN          13000
Raphaely                  PU_MAN          11000
Russell                   SA_MAN          14000
Partners                  SA_MAN          13500
Errazuriz                 SA_MAN          12000
```

OR--returns true if one conditions is ture

```
SQL> select last_name,job_id,salary from employees where salary>10000 OR job_id like '%MAN%';

LAST_NAME                 JOB_ID         SALARY
------------------------- ---------- ----------
Hartstein                 MK_MAN          13000
Higgins                   AC_MGR          12000
King                      AD_PRES         26400
Kochhar                   AD_VP           17000
De Haan                   AD_VP           17000
Greenberg                 FI_MGR          12000
```

NOT--returns true if both conditions are false

```
SQL> select last_name,job_id from employees where job_id not in('IT_PROG','ST_CLERK','SA_REP');

LAST_NAME                 JOB_ID
------------------------- ----------
Baer                      PR_REP
Baida                     PU_CLERK
Bell                      SH_CLERK
Bull                      SH_CLERK
Cabrio                    SH_CLERK
Cambrault                 SA_MAN
```

Note: (parenthesis will be given to give the priority)

9. ORDER BY Clause-

ORDER BY clause will always comes at the last of the select statement.You can have only one order by clause in a single statement.ORDER BY clause is used to sort the data numerically or alphabetically.BY default sorting is ascending.

Sort retrieved rows with the ORDER BY clause: –

-**ASC**: ascending order, default

– **DESC**: descending order

```
SQL> select last_name,hire_date from employees order by hire_date desc;

LAST_NAME                 HIRE_DATE
------------------------- ---------
Banda                     21-APR-00
Kumar                     21-APR-00
Ande                      24-MAR-00
Markle                    08-MAR-00
Lee                       23-FEB-00
Philtanker                06-FEB-00
```

-Sorting by column alias.

```
SQL> SELECT employee_id, last_name, salary*12 annsal FROM   employees ORDER BY annsal;

EMPLOYEE_ID LAST_NAME                    ANNSAL
----------- ------------------------- ----------
        132 Olson                          25200
        136 Philtanker                     26400
        128 Markle                         26400
        127 Landry                         28800
        135 Gee                            28800
```

Sorting by multiple column.

```
SQL> SELECT employee_id, last_name,department_id, salary FROM   employees
  2  order by department_id,salary desc;

EMPLOYEE_ID LAST_NAME                 DEPARTMENT_ID     SALARY
----------- ------------------------- ------------- ----------
        200 Whalen                               10       4400
        201 Hartstein                            20      13000
        202 Fay                                  20       6000
        114 Raphaely                             30      11000
        115 Khoo                                 30       3100
        116 Baida                                30       2900
```

10. -Substitution variable

you can create reports that prompt users to supply their own values to restrict the range of data returned by using substitution variables.

- You can embed substitution variables in a command file or in a single SQL statement.

- A variable can be thought of as a container in which the values are temporarily stored. When the statement is run, the value is substituted.

Use substitution variables to: – Temporarily store values with single-ampersand **(&)** and - double-ampersand (**&&**) substitution. It can be used in

-where condition

-order by clause

-column expressions

-table names

-entire select statements

```
SQL> SELECT employee_id, last_name, salary, department_id FROM   employees
  2  where employee_id=& employee_id;
Enter value for employee_id: 100
old   2: where employee_id=& employee_id
new   2: where employee_id=100

EMPLOYEE_ID LAST_NAME                  SALARY DEPARTMENT_ID
----------- ------------------------- ---------- -------------
        100 King                         26400            90
```

--Character and Date Values--with Substitution Variables.

```
SQL> SELECT last_name, department_id, salary*12 FROM employees WHERE job_id = '&job_title';
Enter value for job_title: SA_REP
old   1: SELECT last_name, department_id, salary*12 FROM employees WHERE job_id = '&job_title'
new   1: SELECT last_name, department_id, salary*12 FROM employees WHERE job_id = 'SA_REP'

LAST_NAME                 DEPARTMENT_ID  SALARY*12
------------------------- ------------- ----------
Tucker                               80     120000
Bernstein                            80     114000
Hall                                 80     108000
Olsen                                80      96000
Cambrault                            80      90000
```

11- && Substitution Variable.

Use the double ampersand (&&) if you want to reuse the variable value without prompting the user each time.

(IT will work same as define .oncce you use && with some variable then for that session that value is assigined to that variable.).

```
SQL> SELECT employee_id, last_name, job_id, &&column_name FROM employees ORDER BY &column_name ;
Enter value for column_name: salary
old   1: SELECT employee_id, last_name, job_id, &&column_name FROM employees ORDER BY &column_name
new   1: SELECT employee_id, last_name, job_id, salary FROM employees ORDER BY salary

EMPLOYEE_ID LAST_NAME                 JOB_ID         SALARY
----------- ------------------------- ---------- ----------
        132 Olson                     ST_CLERK         2100
        136 Philtanker                ST_CLERK         2200
        128 Markle                    ST_CLERK         2200
        127 Landry                    ST_CLERK         2400
        135 Gee                       ST_CLERK         2400
```

12- DEFINE Command.

- Use the DEFINE command to create and assign a value to a variable.

- Use the UNDEFINE command to remove a variable.

```
SQL> DEFINE employee_num = 200
SQL> SELECT employee_id, last_name,salary,department_id FROM employees
  2  where employee_id=&employee_num;
old   2: where employee_id=&employee_num
new   2: where employee_id=200

EMPLOYEE_ID LAST_NAME                      SALARY DEPARTMENT_ID
----------- ------------------------- ---------- -------------
        200 Whalen                          4400            10
```

How to undefine?

```
SQL> UNDEFINE employee_num
```

13- VERIFY Command.

Use the VERIFY command to toggle the display of the substitution variable, both before and after SQL Developer replaces substitution variables with values:

(To confirm the changes in the SQL statement, use the VERIFY command. Setting SET VERIFY ON forces SQL plus to display the text of a command after replaces substitution variables with values.)

```
SQL>  set verify on ;
SQL> show verify
verify ON
SQL>  select last_name,employee_id from employees  where employee_id=&id;
Enter value for id: 100
old   1:  select last_name,employee_id from employees  where employee_id=&id
new   1:  select last_name,employee_id from employees  where employee_id=100

LAST_NAME                 EMPLOYEE_ID
------------------------- -----------
King                              100
```

ASSIGNMENTS.

1. find out the last_name, salary, manager_id of those employees whose salary greater than 10000?
2. create a report who are in department 10,20 or 90 ?
3. select all the employees whose salary between 4000 and 8000?
4. HR required the names and salaries of those employees whose employee id is 101,202,and 203.
5. display the name and job_id who are in department 10 or 50?
6. Display all employee last names in which the second letter of the name is a.
7. Display all employees whose salary is not in 4000,10000,3000 ?
8. Display all the employees whose commssion_pct is null?
9. Display the name, salary, department_id of those employees whose job_id is SA_REP and salary is >4000 ?
10. how to find out the record of different employee with in a one query?
11. Create a report that displays the last name and department number for employee 176.
12. display the last name and salary for any employee whose salary is not in 5000 to 12,000 range.
13. Create a report to display the last name, salary, and commission of all employees who earn commissions. Sort the data in descending order of salary and commissions.
14. Find all the employees whose last name contains e or a chatacter.

5. Functions

5.1 Single Row Function

These functions operate on single rows only and return one result per row. we can perform some task using single row functions like :-

-- Manipulate data items

-- Accept arguments and return one value

-- Act on each row that is returned

-- May modify the data type

--Can be nested

Types of Single row functions :

- Character
- Number
- Date
- Conversion
- General

5.1.1 Character functions.

Accept character input and can return both character and number values.

character functions are two types.

1--Case manupulation functions

2--character manupulation functions

5.1.1.1 CASE MANIPULATION FUNTION

Lower - For converting the string or field into lower case

Examples:

```
SQL> SELECT LOWER('SQL Course') "lower" from dual;

lower
----------
sql course
```

upper - For converting the string or field into upper case

```
SQL> select UPPER('ahmed') from dual;

UPPER
-----
AHMED
```

initcap - For converting the string or field into initcap case

```
SQL> select initcap(' AHMED ') from dual;

INITCAP
-------
 Ahmed
```

5.1.1.2 character manupulation function

These functions are used to perform some actions on string like trimming the string or replacing some character with other or either joining two string.

CONCAT : Joins values together (You are limited to using two parameters with CONCAT.)

```
SQL> select concat('Hello','World') from dual;

CONCAT('HE
----------
HelloWorld

SQL> select concat(last_name,first_name ) from employees;

CONCAT(LAST_NAME,FIRST_NAME)
--------------------------------------------
AbelEllen
AndeSundar
AtkinsonMozhe
AustinDavid
```

SUBSTR: Extracts a string of determined length.i.e you can cut the string in subparts.

```
SQL> select last_name,substr(last_name,1,3) from employees;

LAST_NAME                 SUB
------------------------- ---
Abel                      Abe
Ande                      And
Atkinson                  Atk
Austin                    Aus
Baer                      Bae
Baida                     Bai
Banda                     Ban
```

Substr function having three field .First field is the field on which you are going to perform substr in our case its last_name.Second field is starting position from which you want to start your string.Third Field is optionaml if you don't mention it it will give full string other it will give number of character after starting position.exp.

```
SQL> select last_name,substr(last_name,3) from employees;

LAST_NAME                 SUBSTR(LAST_NAME,3)
------------------------- -----------------------
Abel                      el
Ande                      de
Atkinson                  kinson
Austin                    stin
Baer                      er
```

LENGTH: Shows the total length of a string as a numeric value.

```
SQL> select length('HELLOWORLD') from dual;

LENGTH('HELLOWORLD')
--------------------
                  10

SQL> select last_name,length(last_name) from employees;

LAST_NAME                 LENGTH(LAST_NAME)
------------------------- -----------------
Abel                                      4
Ande                                      4
Atkinson                                  8
Austin                                    6
```

INSTR: Finds the numeric position of a named character(by default it will give the first position of character)

```
SQL> select last_name,instr(last_name,'n') from employees;

LAST_NAME                 INSTR(LAST_NAME,'N')
------------------------- --------------------
Abel                                         0
Ande                                         2
Atkinson                                     5
Austin                                       6
Baer                                         0
```

If you want second occurance of some character then like in this example we will see the second occurance of ‘e’

```
SQL> select last_name,instr(last_name,'n',1,2) from employees
  2  where last_name='Greene';

LAST_NAME                INSTR(LAST_NAME,'N',1,2)
------------------------ ------------------------
Greene                                          0
```

LPAD: Pads the character value right-justified

Syntax:

LPAD(FIELD,TOTAL LENGTH,CHARACTER WITH WHOM YOU WANT TO PAD)

```
SQL> select last_name ,lpad(last_name,10,'*') from employees;

LAST_NAME                 LPAD(LAST_
------------------------- ----------
Abel                      ******Abel
Ande                      ******Ande
Atkinson                  **Atkinson
```

RPAD: Pads the character value left-justified

Syntax:

RPAD(FIELD,TOTAL LENGTH,CHARACTER WITH WHOM YOU WANT TO PAD)

```
SQL> select last_name ,lpad(last_name,10,'*') from employees;

LAST_NAME                 LPAD(LAST_
------------------------- ----------
Abel                      ******Abel
Ande                      ******Ande
Atkinson                  **Atkinson
```

TRIM: Trims heading or trailing characters from a character string (If trim_character or trim_source is a character literal, you must enclose it in single quotation marks.)

```
SQL> select trim(leading 'H' from  'HelloH') from dual;

TRIM(
-----
elloH

SQL> select TRIM('H' FROM 'HelloWorld') from dual;

TRIM('H'F
---------
elloWorld

SQL> select trim('K' from last_name) from employees where last_name='King';

TRIM('K'FROMLAST_NAME)
--------------------------
ing
ing

SQL> select trim(leading 'H' from  'HelloH') from dual;

TRIM(
-----
elloH

SQL> select trim(trailing 'H' from  'HelloH') from dual;

TRIM(
-----
Hello
```

Note : By default trim function will remove H from the both side.

Replace : This function is used to replace any character to some other character.

Syntax:

Replace(column_name,character which you want to replace,character with whom you want to replace).

```
SQL> select last_name,replace(last_name,'b','*') from employees;

LAST_NAME                 REPLACE(LAST_NAME,'B','*'
------------------------- -------------------------
Abel                      A*el
Ande                      Ande
Atkinson                  Atkinson
```

5.1.2 NUMBER Function

Accept numeric input and return numeric values.

- **ROUND**: Rounds value to specified decimal
- **TRUNC**: Truncates value to specified decimal
- **MOD**: Returns remainder of division

```
SQL> SELECT ROUND(45.923,2), ROUND(45.923,0), ROUND(45.923,-1) FROM   DUAL;

ROUND(45.923,2) ROUND(45.923,0) ROUND(45.923,-1)
--------------- --------------- ----------------
          45.92              46               50

SQL> SELECT TRUNC(45.923,2), TRUNC(45.923), TRUNC(45.923,-1) FROM   DUAL;

TRUNC(45.923,2) TRUNC(45.923) TRUNC(45.923,-1)
--------------- ------------- ----------------
          45.92            45               40
```

Example**:(MOD)**

For all employees with job title of Sales Representative,calculate the remainder of the salary after it is divided by 5,000.

```
SQL> SELECT last_name, salary, MOD(salary,5000)FROM employees WHERE job_id = 'SA_REP';

LAST_NAME                    SALARY MOD(SALARY,5000)
------------------------- ---------- ----------------
Tucker                         10000                0
Bernstein                       9500             4500
Hall                            9000             4000
Olsen                           8000             3000
```

5.1.3 Date functions:

Operate on values of the DATE data type (All date functions return a value of DATE data type except the MONTHS_BETWEEN function, which returnsa number.)

-- The Oracle Database stores dates in an internal numeric format: century, year, month, day, hours, minutes, and seconds.

-- The default date display format is DD-MON-RR.

--Enables you to store 21st-century dates in the 20th century by specifying only the last two digits of the year -- Enables you to store 20th-century dates in the 21st century in the same way.

SYSDATE : SYSDATE is a function that returns: Date and Time

```
SQL> select sysdate from dual;

SYSDATE
---------
07-JUL-15
```

Arithmetic with Dates

• Add or subtract a number to or from a date for a resultant date value.

• Subtract two dates to find the number of days between those dates.

```
SQL> select sysdate+3,sysdate-3 from dual;

SYSDATE+3 SYSDATE-3
--------- ---------
10-JUL-15 04-JUL-15
```

(it will adding 3 days and subtracting 3 days in the date.similarly difference between two date will give no of days.)

```
SQL>  SELECT last_name, (SYSDATE-hire_date)/7 AS WEEKS FROM employees WHERE department_id=90;

LAST_NAME                      WEEKS
------------------------- ----------
King                        1463.9549
Kochhar                    1345.81204
De Haan                     1172.9549
```

The abovestatement will give the total number of week.

Date Functions

Function	Result
MONTHS_BETWEEN	Number of months between two dates
ADD_MONTHS	Add calendar months to date
NEXT_DAY	Next day of the date specified
LAST_DAY	Last day of themonth
ROUND	Round date with month or either year
TRUNC	Truncate date with month or either year

MONTHS_BETWEEN:

This function will gives total number of month between two date.

MONTHS_BETWEEN(date1,date2).

```
SQL> select months_between(sysdate,hire_date) from employees;

MONTHS_BETWEEN(SYSDATE,HIRE_DATE)
---------------------------------
                       192.570575
                        185.82864
                       333.699608
```

ADD_MONTHS:

This function will add number of months to the given date.

ADD_MONTHS(date1,no of month you want to add)

```
SQL> select sysdate,add_months(sysdate,6) from dual;

SYSDATE   ADD_MONTH
--------- ---------
07-JUL-15 07-JAN-16
```

LAST_DAY:

This Function will gives the last date of the month.

LAST_DAY(date1)

```
SQL> select sysdate,last_day(sysdate) from dual;

SYSDATE   LAST_DAY(
--------- ---------
07-JUL-15 31-JUL-15
```

NEXT_DAY:

This function will give the date of upcoming day.

NEXT_DAY(sysdate,upcoming day name whose date you wanna know)

```
SQL> select sysdate,next_day(sysdate,'friday') from dual;

SYSDATE   NEXT_DAY(
--------- ---------
07-JUL-15 10-JUL-15
```

ROUND

```
SQL> select ROUND(SYSDATE,'MONTH'),ROUND(SYSDATE,'YEAR') from dual;

ROUND(SYS ROUND(SYS
--------- ---------
01-JUL-15 01-JAN-16
```

Note :- if date value between 1 to 15 then it will return u 1st day of the same month if date value is greater then 15 then it will return you 1st day of the next month.

Note :- If month value varies between jan to june then it will return u 1st Jan of the same year if month value is july to dec then it will return u 1st Jan of the next year.

TRUNC

```
SQL> select TRUNC(SYSDATE ,'YEAR'),TRUNC(SYSDATE,'MONTH') from dual;

TRUNC(SYS TRUNC(SYS
--------- ---------
01-JAN-15 01-JUL-15
```

Note: Trunc function always return u 1st day of the same month in the case of month and in the case of year it will return u 1st jan of the same year.

5.1.4 Conversion functions

Convert a value from one data type to another

There are two Data type conversion

1-Implicit data type conversion

2-Implicit data type conversion

-Implicit Data Type Conversion

For assignments, the Oracle server can automatically convert the following:

From	To
VARCHAR2 or CHAR	NUMBER
VARCHAR2 or CHAR	DATE
NUMBER	Varchar2
DATE	Varchar2

Explicit Data Type Conversion

--To_number

--To_Date

--To_char

TO_CHAR Function with Dates :

TO_CHAR(date, 'format_model')

The format model:

- Must be enclosed by single quotation marks
- Is case sensitive
- Can include any valid date format element
- Has an fm element to remove padded blanks or suppress leading zeros
- Is separated from the date value by a comma

```
SQL>  SELECT employee_id, TO_CHAR(hire_date, 'MM/YY') Month_Hired FROM   employees
  2  where last_name='Higgins';

EMPLOYEE_ID MONTH
----------- -----
        205 06/94
```

```
SQL> SELECT last_name,TO_CHAR(hire_date, 'fmDD Month YYYY') AS HIREDATE FROM   employees;

LAST_NAME                 HIREDATE
------------------------- ------------------
OConnell                  21 June 1999
Grant                     13 January 2000
Whalen                    17 September 1987
```

TO_CHAR Function with Numbers

```
SQL> SELECT TO_CHAR(salary, '$99,999.00') SALARY FROM   employees WHERE last_name = 'Ernst';

SALARY
-----------
  $6,000.00
```

TO_NUMBER and TO_DATE Functions

Convert a character string to a number format using the TO_NUMBER function:

TO_NUMBER(char[, 'format_model'])

```
SQL> select * from employees where salary > to_number('$10,000.00','$99,999.99');

EMPLOYEE_ID FIRST_NAME           LAST_NAME
----------- -------------------- -------------------------
EMAIL                     PHONE_NUMBER         HIRE_DATE JOB_ID         SALARY
------------------------- -------------------- --------- ---------- ----------
COMMISSION_PCT MANAGER_ID DEPARTMENT_ID
-------------- ---------- -------------
        201 Michael              Hartstein
MHARTSTE                  515.123.5555         17-FEB-96 MK_MAN          13000
                      100             20
```

Convert a character string to a date format using the TO_DATE function

Syntax:

TO_DATE(char[, 'format_model'])

```
SQL> select employee_id,hire_date from employees where hire_date= to_date('FEB 17 1996','month dd yyyy');

EMPLOYEE_ID HIRE_DATE
----------- ---------
        201 17-FEB-96
```

- **Single-row functions can be nested to any level.**
- **Nested functions are evaluated from the deepest level to the least deep level.**

```
SQL> SELECT last_name,UPPER(CONCAT(SUBSTR(LAST_NAME,1,3),'_US'))FROM employees WHERE department_id = 60;

LAST_NAME                 UPPER(
------------------------- ------
Hunold                    HUN_US
Ernst                     ERN_US
```

5.1.5 GENERAL Function

- NVL (expr1, expr2)

- NVL2 (expr1, expr2, expr3)

- NULLIF (expr1, expr2)

- COALESCE (expr1, expr2, ..., exprn)

- CASE – DECODE

NVL Function :

Converts a null value to an actual value:

• Data types that can be used are date, character, and number. • Data types must match:

– NVL(commission_pct,0)

– NVL(hire_date,'01-JAN-97')

– NVL(job_id,'No Job Yet')

```
SQL> select employee_id,commission_pct,nvl(commission_pct,0) from employees;

EMPLOYEE_ID COMMISSION_PCT NVL(COMMISSION_PCT,0)
----------- -------------- ---------------------
        198                                   0
        199                                   0
        200                                   0
```

NVL2 Function :

The NVL2 function examines the first expression. If the first expression is not null, then the NVL2 returns the second expression. If the first expression is null, then the third expression is returned.

Syntax:

NVL2(expr1, expr2, expr3)

```
SQL> SELECT last_name,salary,commission_pct,NVL2(commission_pct,'SAL+COMM','SAL')income FROM
  2  employees where department_id in (20,50);

LAST_NAME                     SALARY COMMISSION_PCT INCOME
------------------------- ---------- -------------- --------
Hartstein                      13000                SAL
Fay                             6000                SAL
OConnell                        2600                SAL
```

NULLIF Function:

The NULLIF function compares two expressions. If they are equal, the function returns null. If they not equal, the function returns the first expression. You cannot specify the literal NULL for the first expression.

Syntax: NULLIF (expr1, expr2)

```
SQL> SELECT first_name, LENGTH(first_name) "expr1", last_name, LENGTH(last_name) "expr2",
  2   NULLIF(LENGTH(first_name), LENGTH(last_name)) result FROM   employees;

FIRST_NAME                expr1 LAST_NAME                     expr2     RESULT
-------------------- ---------- ------------------------- ---------- ----------
Ellen                         5 Abel                               4          5
Sundar                        6 Ande                               4          6
Mozhe                         5 Atkinson                           8          5
```

Using the COALESCE Function

The COALESCE function returns the first non-null expression in the list.

Syntax COALESCE (expr1, expr2, ... exprn)

```
SQL> SELECT last_name,COALESCE(manager_id,commission_pct,-1)FROM employees ORDER BY commission_pct;

LAST_NAME                 COALESCE(MANAGER_ID,COMMISSION_PCT,-1)
------------------------- --------------------------------------
Lee                                                          147
Johnson                                                      149
Marvins                                                      147
```

Conditional Expressions

- **Provide the use of IF-THEN-ELSE logic within a SQL statement**
- **Use two methods:**

-CASE expression

– DECODE function

CASE Expression : Facilitates conditional inquiries by doing the work of an IF-THEN-ELSE statement.

Example : If JOB_ID is IT_PROG, the salary increase is 10%; if JOB_ID is ST_CLERK, the salary increase is 15%; if JOB_ID is SA_REP, the salary increase is 20%. For all other job roles, there is no increase in salary?

```
SQL>  SELECT last_name, job_id, salary, CASE job_id WHEN 'IT_PROG' THEN 1.10*salary
  2   WHEN 'ST_CLERK' THEN 1.15*salary WHEN 'SA_REP'    THEN 1.20*salary
  3  ELSE      salary END      "REVISED_SALARY" FROM   employees;

LAST_NAME                 JOB_ID         SALARY REVISED_SALARY
------------------------- ---------- ---------- --------------
OConnell                  SH_CLERK         2600           2600
Grant                     SH_CLERK         2600           2600
Whalen                    AD_ASST          4400           4400
Hartstein                 MK_MAN          13000          13000
```

The same statement can be written with the DECODE function Using the DECODE Function.

```
SQL> SELECT last_name, job_id, salary, DECODE(job_id, 'IT_PROG', 1.10*salary,
  2  'ST_CLERK', 1.15*salary, 'SA_REP',   1.20*salary, salary) REVISED_SALARY
  3   FROM   employees;

LAST_NAME                 JOB_ID         SALARY REVISED_SALARY
------------------------- ---------- ---------- --------------
OConnell                  SH_CLERK         2600           2600
Grant                     SH_CLERK         2600           2600
Whalen                    AD_ASST          4400           4400
```

Questions:

1. Create a report that displays all the employees and indicate with the words Yes or No whether they receive a commission. Use the DECODE or NVL2 expression in your query.

2. How we will get this output?

   Steven.king@focus.com

3. Display the OUTPUT: The job id for KING is ad_pres

4. replace AA where a in the last_name.

5. Display the OUTPUT:

   AWalsh@focus.com

   MWeiss@focus.com

   JWhalen@focus.com

5.2 MULTIPLE ROW/GROUP function

Group functions operate on sets of rows to give one result per group.

Types of Group Functions

- AVG
- SUM
- COUNT
- MAX
- MIN

Using the AVG Functions

You can use AVG for numeric data.

```
SQL> SELECT AVG(SALARY) FROM EMPLOYEES;

AVG(SALARY)
-----------
 6484.14019
```

Using the SUM ,MIN,MAX Functions

```
SQL> SELECT SUM(SALARY),MIN(SALARY),MAX(SALARY) FROM EMPLOYEES;

SUM(SALARY) MIN(SALARY) MAX(SALARY)
----------- ----------- -----------
     693803        2100       26400
```

Using the COUNT Function

COUNT Function has three format:

COUNT(*)

```
SQL> SELECT COUNT(*) FROM EMPLOYEES;

  COUNT(*)
----------
       107
```

COUNT(expr)

```
SQL>  SELECT COUNT(commission_pct)  FROM   employees  WHERE department_id = 50;

COUNT(COMMISSION_PCT)
---------------------
                    0
```

COUNT(DISTINCT expr) FOR UNIQUE VALUES.

```
SQL> SELECT COUNT(DISTINCT department_id) FROM   employees;

COUNT(DISTINCTDEPARTMENT_ID)
----------------------------
                          11
```

NOTE:: Group functions ignore null values in the column. The NVL function forces group functions to include null values.

```
SQL> SELECT AVG(nvl(commission_pct,0)) FROM   employees;

AVG(NVL(COMMISSION_PCT,0))
--------------------------
                .072897196
```

GROUP BY Clause:

You can divide rows in a table into smaller groups by using the GROUP BY clause.

```
SQL> SELECT   department_id, AVG(salary) FROM employees GROUP BY department_id ;

DEPARTMENT_ID AVG(SALARY)
------------- -----------
          100        8600
           30        4150
                     7000
```

NOTE::All columns in the SELECT list that are not in group functions must be in the GROUP BY clause.

```
SQL> SELECT department_id, COUNT(last_name) FROM   employees;
SELECT department_id, COUNT(last_name) FROM   employees
       *
ERROR at line 1:
ORA-00937: not a single-group group function
```

you can also use that column which is not in SELECT list LIKE

```
SQL>  SELECT   AVG(salary) FROM employees GROUP BY department_id ;

AVG(SALARY)
-----------
       8600
       4150
       7000
```

Grouping by More Than One Column

```
SQL> SELECT department_iD, job_id,SUM(salary) FROM employees GROUP BY department_id,job_id ;

DEPARTMENT_ID JOB_ID     SUM(SALARY)
------------- ---------- -----------
          110 AC_ACCOUNT         8300
           90 AD_VP             34000
           50 ST_CLERK          55700
           80 SA_REP           243500
```

HAVING Clause:

Restricting Group Results with the HAVING Clause

-You cannot use the WHERE clause to restrict groups.

-You use the HAVING clause to restrict groups.

-You cannot use group functions in the WHERE clause

```
SQL> SELECT department_id,AVG(salary) FROM employees WHERE AVG(salary)>8000 GROUP BY department_id;
SELECT department_id,AVG(salary) FROM employees WHERE AVG(salary)>8000 GROUP BY department_id
                                                      *
ERROR at line 1:
ORA-00934: group function is not allowed here
```

SO TO AVOID THIS ERROR WE HAVE TO USE HAVING CLAUSE.

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

DEPARTMENT_ID AVG(SALARY)
------------- -----------
          100        8600
           20        9500
           70       10000
```

-Nesting Group Functions

```
SQL> SELECT   MAX(AVG(salary)) FROM     employees GROUP BY department_id;

MAX(AVG(SALARY))
----------------
      20133.3333
```

Note: Group functions can be nested to a depth of two. You can not user a column with nested group function.

GROUP BY with ROLLUP and CUBE Operators

If i have to find out the total salary of employees then i write this statement.

```
SQL> select sum(salary) from employees;

SUM(SALARY)
-----------
     693803
```

If i have to find out the department wise sum salary then

```
SQL> select department_id,sum(salary) from employees group by department_id;

DEPARTMENT_ID SUM(SALARY)
------------- -----------
          100       51600
           30       24900
                     7000
```

If i have to find out the department wise and job wise sum salary then

```
SQL> select department_id,job_id,sum(salary) from employees group by department_id,job_id;

DEPARTMENT_ID JOB_ID     SUM(SALARY)
------------- ---------- -----------
          110 AC_ACCOUNT        8300
           90 AD_VP            34000
           50 ST_CLERK         55700
           80 SA_REP          243500
```

Using Rollup/Cube i can get these three different results in a single query.

ROLLUP Operator

- ROLLUP is an extension to the GROUP BY clause.
- Use the ROLLUP operation to produce subtotal and grand total of aggregated rows.
- Based on (n=n+1). n=no.columns n+1= no of combinations.

Syntax: SELECT [column,] group_function(column). . .

FROM table

[WHERE condition]

[GROUP BY [ROLLUP] group_by_expression]

[HAVING having_expression]

[ORDER BY column];

```
SQL> SELECT   department_id, job_id, SUM(salary) FROM   employees
  2  WHERe department_id<30 GROUP BY ROLLUP(department_id, job_id);

DEPARTMENT_ID JOB_ID     SUM(SALARY)
------------- ---------- -----------
           10 AD_ASST           4400
           10                   4400
           20 MK_MAN           13000
           20 MK_REP            6000
           20                  19000
                               23400
```

CUBE Operator

- CUBE is an extension to the GROUP BY clause.
- You can use the CUBE operator to produce cross- tabulation values with a single SELECT statement.
- Based on n=2n

Syntax: SELECT [column,] group_function(column). . .

FROM table

[WHERE condition]

[GROUP BY [CUBE] group_by_expression]

[HAVING having_expression]

[ORDER BY column];

```
SQL> SELECT   department_id, job_id, SUM(salary) FROM  employees
  2  where department_id<20  GROUP BY CUBE (department_id, job_id) ;

DEPARTMENT_ID JOB_ID     SUM(SALARY)
------------- ---------- -----------
                                4400
              AD_ASST           4400
           10                   4400
           10 AD_ASST           4400
```

5.3 ANALYTICAL FUNCTION

5.3.1 LISTAGG FUNCTION

The Oracle/PLSQL LISTAGG function concatenates values of the *measure_column* for each GROUP based on the *order_by_clause*

Syntax:

```
LISTAGG (measure_column [, 'delimiter'])
  WITHIN GROUP (order_by_clause) [OVER (query_partition_clause)]
```

Parameters or Arguments

measure_column

The column whose values you wish to concatenate together in the result set. Null values in the *measure_column* are ignored.

delimiter

Optional. It is the delimiter to use when separating the *measure_column* values when outputting the results.

order_by_clause

It determines the order that the concatenated values (ie: *measure_column*) are returned.

```
SQL> select department_id,listagg(employee_id,'/') within group (order by employee_id)
  2  "SAME DEPARTMENT" from employees group by department_id;

DEPARTMENT_ID SAME DEPARTMENT
------------- --------------------------------------------
           10 200
           20 201/202
           30 114/115/116/117/118/119
           40 203
           50 120/121/122/123/124/125/126/127/128/129/130/1
              31/132/133/134/135/136/137/138/139/140/141/14
              2/143/144/180/181/182/183/184/185/186/187/188
              /189/190/191/192/193/194/195/196/197/198/199
```

5.3.2 RANK FUNCTION

The Oracle/PLSQL RANK function returns the rank of a value in a group of values. It is very similar to the DENSE_RANK function. However, the rank function can cause non-consecutive rankings if the tested values are the same. Whereas, the DENSE_RANK function will always result in consecutive rankings.

Syntax.

```
rank() OVER ( [ query_partition_clause] ORDER BY clause )
```

Let's assume we want to assign a sequential order, or rank, to people within a department based on salary, we might use the RANK function like.

```
SELECT empno,
       deptno,
       sal,
       RANK() OVER (PARTITION BY deptno ORDER BY sal) "rank"
FROM   emp;

     EMPNO     DEPTNO        SAL       rank
---------- ---------- ---------- ----------
      7934         10       1300          1
      7782         10       2450          2
      7839         10       5000          3
      7369         20        800          1
      7876         20       1100          2
      7566         20       2975          3
      7788         20       3000          4
      7902         20       3000          4
      7900         30        950          1
      7654         30       1250          2
      7521         30       1250          2
      7844         30       1500          4
      7499         30       1600          5
      7698         30       2850          6
```

What we see here is where two people have the same salary they are assigned the same rank. When multiple rows share the same rank the next rank in the sequence is not consecutive.

5.3.3 DENSE RANK

The DENSE_RANK function acts like the RANK function except that it assigns consecutive ranks

```
SELECT empno,
       deptno,
       sal,
       DENSE_RANK() OVER (PARTITION BY deptno ORDER BY sal) "rank"
FROM   emp;
```

EMPNO	DEPTNO	SAL	rank
7934	10	1300	1
7782	10	2450	2
7839	10	5000	3
7369	20	800	1
7876	20	1100	2
7566	20	2975	3
7788	20	3000	4
7902	20	3000	4
7900	30	950	1
7654	**30**	**1250**	**2**
7521	**30**	**1250**	**2**
7844	**30**	**1500**	**3**
7499	30	1600	4
7698	30	2850	5

Questions:

1. Write a query to display the number of people with the same job.
2. Find the difference between the highest and the lowest salaries. Label the column DIFFERENCE.
3. Create a report to display the manager number and the salary of the lowest-paid employee for that manager. Exclude anyone whose manager is not known. Exclude any groups where the minimum salary is $6,000 or less. Sort the output in descending order of salary.
4. Write a query to display the number of people with the same job.
5. Display the manager no and the salary of the lowest-paid employee for that manager. Exclude anyone whose manager is not known. Exclude any groups where the minimum salary is 6000 or less. Sort the output in desc order of salary.

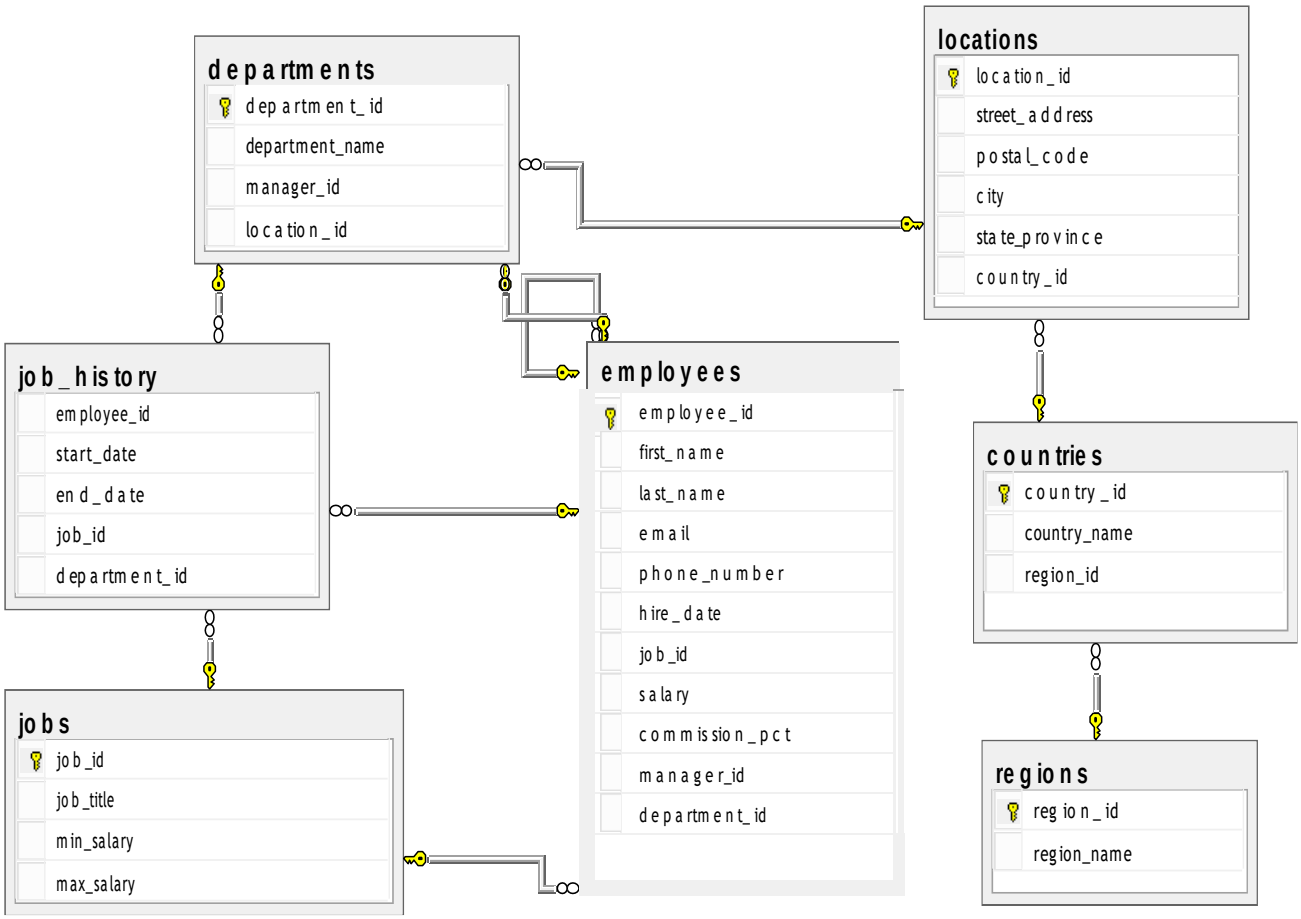
6. JOIN

Sometime you need to use data from two separate tables for that you need to join wo table.

- Employee IDs exist in the EMPLOYEES table.
- Department IDs exist in both the EMPLOYEES and DEPARTMENTS tables.
- Department names exist in the DEPARTMENTS table.

To produce the report, you need to link the EMPLOYEES and DEPARTMENTS tables and access data.

Diagram For HR database Model



Types of Joins

- Cartetian Products
- Cross joins
- Natural joins
- USING clause
- Full (or two-sided) outer joins
- Arbitrary join conditions for outer joins.

6.1 CARTESIAN PRODUCTS

- A Cartesian product is performed when:

– A join condition is omitted

– A join condition isinvalid

– All rows in the first table are joined to all rows in the second table

- To avoid a Cartesian product, always include a valid join condition.

SELECT e.employee_id,d.department_id,d.department_name FROM employees e, departments d;

```
EMPLOYEE_ID DEPARTMENT_ID DEPARTMENT_NAME
----------- ------------- ------------------------------
        200           270 Payroll
        201           270 Payroll
        202           270 Payroll
        203           270 Payroll
        204           270 Payroll
        205           270 Payroll
        206           270 Payroll

2889 rows selected.
```

6.2 -Creating Cross Joins

The CROSS JOIN clause produces the cross-product of two tables.

- This is also called a Cartesian product between the two tables.

SELECT last_name, department_name FROM employees CROSS JOIN departments.

```
LAST_NAME                 DEPARTMENT_NAME
------------------------- ------------------------------
Vargas                    Payroll
Vishney                   Payroll
Vollman                   Payroll
Walsh                     Payroll
Weiss                     Payroll
Whalen                    Payroll
Zlotkey                   Payroll

2889 rows selected.
```

6.3 Creating Natural Joins

- The NATURAL JOIN clause is based on all columns in the two tables that have the same name.
- It selects rows from the two tables that have equal values in all matched columns.
- If the columns having the same names have different data types, an error is returned.

```
SQL> SELECT  department_id,location_id, city FROM departments NATURAL JOIN locations ;

DEPARTMENT_ID LOCATION_ID CITY
------------- ----------- ------------------------------
           60        1400 Southlake
           50        1500 South San Francisco
           10        1700 Seattle
```

NOTE :natural join will give less number of row if it is having more than one identical column.

```
SQL> SELECT  department_id,LOCATION_ID, CITY from DEPARTMENTS nATURAL JOIN
  2   locations WHERE department_id IN (20,50);

DEPARTMENT_ID LOCATION_ID CITY
------------- ----------- ------------------------------
           50        1500 South San Francisco
           20        1800 Toronto
```

6.4 Creating Joins with the USING Clause

- Use the USING clause to match only one column when more than one column matches.
- Do not use a table name or alias in the referenced columns.

```
SQL> SELECT  employees.employee_id, employees.last_name,
  2  departments.location_id, department_id
  3  FROM   employees JOIN departments USING (department_id) ;

EMPLOYEE_ID LAST_NAME                 LOCATION_ID DEPARTMENT_ID
----------- ------------------------- ----------- -------------
        200 Whalen                           1700            10
        201 Hartstein                        1800            20
        202 Fay                              1800            20
```

6.5 Creating Joins with the ON Clause

- Use table aliases to simplify queries.
- Use table aliases to improve performance.

Use ON clause if column name in both the table is different.Or You want to self join the table.

```
SQL>  SELECT e.last_name emp, m.last_name mgr FROM   employees e
  2   JOIN employees m ON    (e.manager_id = m.employee_id);

EMP                       MGR
------------------------- -------------------------
Smith                     Cambrault
Ozer                      Cambrault
Kumar                     Cambrault
```

6.6 Applying Additional Conditions to a Join

For Additional restriction on desired output or for specific out put we apply some restriction in where clause.

```
SQL> SELECT e.employee_id, e.last_name, e.department_id,
  2  d.department_id, d.location_id FROM   employees e JOIN departments d
  3  ON     (e.department_id = d.department_id) AND    e.manager_id = 149 ;

EMPLOYEE_ID LAST_NAME                 DEPARTMENT_ID DEPARTMENT_ID LOCATION_ID
----------- ------------------------- ------------- ------------- -----------
        174 Abel                                80            80        2500
        175 Hutton                              80            80        2500
        179 Johnson                             80            80        2500
```

6.7 Creating Three-Way Joins with the ON Clause

Some time you need to join more than two table .you can do it by n way join .

```
SQL>  SELECT employee_id, city, department_name FROM   employees e JOIN   departments d
  2   ON     d.department_id = e.department_id JOIN   locations l
  3   ON     d.location_id = l.location_id;

EMPLOYEE_ID CITY                          DEPARTMENT_NAME
----------- ----------------------------- ------------------------------
        100 Seattle                       Executive
        101 Seattle                       Executive
        102 Seattle                       Executive
        103 Southlake                     IT
```

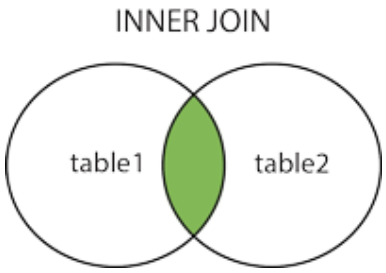
6.8 Non equijoins

A non equijoin is join condition containing something other than equality = operator.

```
SQL> SELECT e.last_name, e.salary, j.job_title FROM   employees e JOIN jobs j
  2   ON     e.salary  BETWEEN 3000 AND 6000;

LAST_NAME                    SALARY JOB_TITLE
------------------------- ---------- -----------------------------------
Whalen                         4400 President
Fay                            6000 President
Ernst                          6000 President
```

6.9 Inner joins

The INNER JOIN keyword selects all rows from both tables as long as there is a match between the columns in both tables.inner join is same as join.



6.10 Outer join

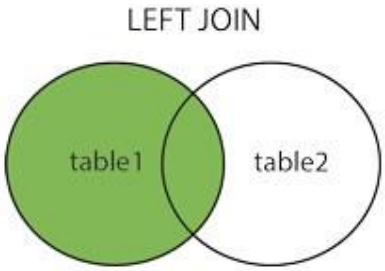
There are three types of outer joins:

- LEFT OUTER JOIN
- RIGHT OUTER JOIN
- FULL OUTER JOIN

6.10.1 Left Outer Join

The LEFT JOIN keyword returns all rows from the left table (table1), with the matching rows in the right table (table2). The result is NULL in the right side when there is no match.

```
SELECT column_name(s)
FROM table1
LEFT OUTER JOIN table2
ON table1.column_name=table2.column_name;
```



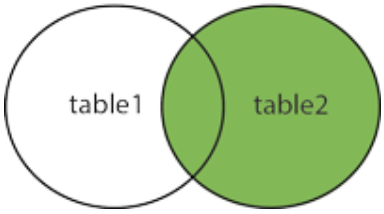

```
SQL>  SELECT e.last_name, e.department_id, d.department_name FROM
  2  employees e LEFT OUTER JOIN departments d ON   (e.department_id = d.department_id) ;

LAST_NAME                 DEPARTMENT_ID DEPARTMENT_NAME
------------------------- ------------- ------------------------------
Whalen                               10 Administration
Fay                                  20 Marketing
Hartstein                            20 Marketing
```

6.10.2 Right Outer Join

The RIGHT JOIN keyword returns all rows from the right table (table2), with the matching rows in the left table (table1). The result is NULL in the left side when there is no match.

```
SELECT column_name(s)
FROM table1
RIGHT OUTER JOIN table2
ON table1.column_name=table2.column_name;
```

RIGHT JOIN




```
SQL> SELECT e.last_name, e.department_id, d.department_name FROM
  2   employees e RIGHT OUTER JOIN departments d ON    (e.department_id = d.department_id)
  3  order by 3 desc;

LAST_NAME                 DEPARTMENT_ID DEPARTMENT_NAME
------------------------- ------------- ------------------------------
                                        Treasury
Feeney                               50 Shipping
Sullivan                             50 Shipping
Fleaur                               50 Shipping
```

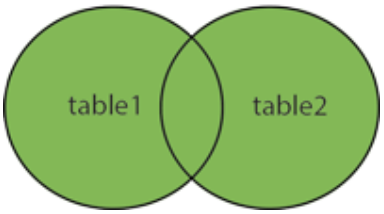
6.10.3 Full Outer Join

The FULL OUTER JOIN keyword returns all rows from the left table (table1) and from the right table (table2).

The FULL OUTER JOIN keyword combines the result of both LEFT and RIGHT joins.

```
SELECT column_name(s)
FROM table1
FULL                                    OUTER                                    JOIN table2
ON table1.column_name=table2.column_name;
```

FULL OUTER JOIN




```
SQL> SELECT e.last_name, d.department_id, d.department_name FROM
  2  employees e FULL OUTER JOIN departments d ON    (e.department_id = d.department_id)
  3  order by 2 desc;

LAST_NAME                 DEPARTMENT_ID DEPARTMENT_NAME
------------------------- ------------- ------------------------------
Grant
                                   270 Payroll
                                   260 Recruiting
                                   250 Retail Sales
```

Questions:

1. Write a query for the HR department to produce the addresses of all the departments. Use the LOCATIONS and COUNTRIES tables. Show the location ID, street address, city, state and country in the output. Use a NATURAL JOIN to produce the results.
2. Write a query to display the last name,department number, and department name for all employees.
3. Display the last name, job, department number, and department name for all employees who work in Toronto.

Equi join

4. Create a report that displays the department name, location, name, job title, and salary of those employees who work in a specific location. Prompt the user for the location. For example, if the user enters 1800, the following are the results:
5. Create a report that displays the jobs that are found in the Administration and Executive departments.
6. Display the employee name and their country .
7. Write a query to display last_name, job_id, department_name who works in Toronto.

Self Join

8. write a query to display the employee name and their manager's name.
9. Write a query to find the names and hire dates for all employees who were hired before their managers, along with their managers name and hire date.

Natural Join

10. Write a query to display the location id, street address, city, country name.

Using Clause 1.Write a query to display the last_name, department_name, department_id using using clause.

Left Outer Join

11. Display employees's name and the department even if there is no department assigned to the employee.

2.write a query to display the employee name and their manager's name. display all employees including king who has no manager.

Right Outer Join

12. Display employees's name and the department. List all departments even if there is no employee assigned to them.

Full Outer Join

13. Display employees's name and the department name .retrieves all rows in the employees table even if there is no match in the departments table, and retrieves all rows in the departments table even if there is no match in the employees table

Joins With Group Functions –

14. Display the country name and total no of employees in that country.
15. display the jobs that are found in the Administration and Executive departments.Also display the no of employees for these jobs. Show the job with the highest no of employee first.

7. SubQuery

A Subquery or Inner query or Nested query is a query within another SQL query and embedded within the WHERE clause.

A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.

Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN. etc

You can place the subquery in a number of SQL clauses, including the following:

- WHERE clause
- HAVING clause
- FROM clause

```
SQL> SELECT last_name, salary FROM   employees WHERE salary >
  2  (SELECT salary FROM    employees WHERE last_name = 'Abel');

LAST_NAME                      SALARY
------------------------- ----------
Hartstein                       13000
Higgins                         12000
King                            26400
```

-Types of Subqueries

- Single-row subquery
- Multiple-row subquery
- Scalar subquery
- Multiple-Column subquery
- Correlated Subquery
- Inline View

7.1 Single Row Subquery

Subqueries that can return only one or zero rows to the outer statement are called **single-row subqueries**. Single-row subqueries are subqueries used with a comparison operator in a WHERE, or HAVING clause.

- Use single-row comparison operators

>

<

=

<>

>=

<=

Example:: -Display the employees whose job ID is the same as that of employee 141.?

```
SQL> SELECT last_name, job_id  FROM employees where job_id=
  2  (SELECT job_id FROM      employees  WHERE employee_id = 141
  3  );

LAST_NAME                 JOB_ID
------------------------- ----------
Nayer                     ST_CLERK
Mikkilineni               ST_CLERK
Landry                    ST_CLERK
```

Example:: Display the employees whose job_id id the same as that of employee 141 and whose salary is greater than that of employee 143

```
SQL>  SELECT last_name, job_id, salary FROM  employees WHERE job_id =
  2  (SELECT job_id FROM   employees WHERE employee_id =141) and salary >
  3  (SELECT salary  FROM   employees WHERE employee_id = 143);

LAST_NAME                 JOB_ID         SALARY
------------------------- ---------- ----------
Nayer                     ST_CLERK         3200
Mikkilineni               ST_CLERK         2700
Bissot                    ST_CLERK         3300
```

-Using Group Functions in a Subquery

Example:: Display the employee last name , job id and salary of all employees whose salary is equal to the minimum salary.?

```
SQL> SELECT last_name, job_id, salary FROM   employees WHERE salary =
  2  (SELECT MIN(salary)                 FROM   employees);

LAST_NAME                 JOB_ID         SALARY
------------------------- ---------- ----------
Olson                     ST_CLERK         2100
```

-The HAVING Clause with Subqueries

Example:: Display all the departments that have a minimum salary greater than that of department 50.??

```
SQL> SELECT   department_id, MIN(salary) FROM employees GROUP BY department_id HAVING
  2  min(salary)> (select min(salary)  FROM   employees  WHERE department_id = 50);

DEPARTMENT_ID MIN(SALARY)
------------- -----------
          100        6900
           30        2500
                     7000
```

7.2 Multiple Row subQuery

Subqueries that can return more than one row (but only one column) to the outer statement are called **multiple-row subqueries**. Multiple-row subqueries are subqueries used with an IN, ANY, or ALL clause.

Use multiple-row comparison operators.

--IN-- Equal to any member in the list

--ANY-- compare value to each value returned by the subquery

--ALL--compare value to every value returned by the subquery

```
SQL> SELECT last_name, job_id, salary FROM   employees WHERE salary IN
  2   (SELECT salary FROM   employees where department_id in(90,50));

LAST_NAME                 JOB_ID         SALARY
------------------------- ---------- ----------
Matos                     ST_CLERK         2600
Himuro                    PU_CLERK         2600
Grant                     SH_CLERK         2600
OConnell                  SH_CLERK         2600
```

Example: using ANY operator

>ANY greater than the minimum value

<ANY less than the maximum value)

Suppose your inner query gives out put of salary 3000,4000 and 5000.If you use >any then it will all the values which are greate than 3000 or else if you use <any then it will giveall the data which are less than 5000.

```
SQL> select employee_id,salary from employees
  2  where salary<any(3000,4000,5000);

EMPLOYEE_ID     SALARY
----------- ----------
        198       2600
        199       2600
        200       4400
        105       4800
        106       4800
```

Here in the above example we manually given the values so that we can understand the concept of <any or >any.

```
SQL> select employee_id,salary from employees
  2  where salary>any(select salary from employees where department_id=20);

EMPLOYEE_ID     SALARY
----------- ----------
        100      26400
        101      17000
        102      17000
        145      14000
```

Example: using ALL operator

>ALL greater than the maximum value

<ALL less than the minimum value

```
SQL> select employee_id,salary from employees
  2  where salary<all(3000,4000,5000);

EMPLOYEE_ID     SALARY
----------- ----------
        198       2600
        199       2600
        116       2900
        117       2800
        118       2600
```

```
SQL> select employee_id,salary from employees
  2  where salary>all(select salary from employees where department_id=20);

EMPLOYEE_ID     SALARY
----------- ----------
        146      13500
        145      14000
        102      17000
```

7.3 Scalar Subquery

If your inner query return exactly one row n one column then we call it a Scalar Subquery.

```
SQL>  SELECT last_name, job_id, salary FROM   employees WHERE salary =
  2  (SELECT MIN(salary)  FROM   employees);

LAST_NAME                 JOB_ID         SALARY
------------------------- ---------- ----------
Olson                     ST_CLERK         2100
```

7.4 Multiple Column Subquery

If you want to compare two or more column you must write a compound WHERE clause using logical operator.

Column Comparisons

Multiple-column comparisons involving subqueries can be:

- Nonpairwise comparisons
- Pairwise comparisons

7.4.1 Pairwise Comparison Subquery

Example:

-Display the details of the employees who work in the same department , and have the same manager as ‘John’?

```
SQL> select employee_id,manager_id,department_id from employees where
  2  (manager_id,department_id) IN (select manager_id, department_id
  3  from employees where first_name='John');

EMPLOYEE_ID MANAGER_ID DEPARTMENT_ID
----------- ---------- -------------
        137        123            50
        138        123            50
        139        123            50
        140        123            50
```

7.4.2 Nonpairwise Comparison Subquery

Example:

-Display the details of the employees who work in the same department , and have the same manager as 'John'?

```
SQL> select last_name,manager_id,department_id from employees where manager_id IN
  2  (select manager_id   from employees   where first_name='John') AND
  3  department_id IN (select department_id  from employees  where first_name ='John');

LAST_NAME                 MANAGER_ID DEPARTMENT_ID
------------------------- ---------- -------------
Weiss                            100            50
Fripp                            100            50
Kaufling                         100            50
Vollman                          100            50
Mourgos                          100            50
```

7.5 Correlated Subqueries

Correlated subqueries are used for row-by-row processing. Each subquery is executed once for every row of the outer query.

Syntax:

SELECT column1, column2, . .. FROM

table1 outer WHERE column1 operator

(SELECT column1, column2 FROM table2

WHERE expr1 = outer.expr2);

Example: Find all employees who earn more than the average salary in their department.

```
SQL>  SELECT last_name, salary, department_id FROM   employees outer WHERE salary >
  2   (SELECT AVG(salary) FROM   employees WHERE department_id = outer.department_id);

LAST_NAME                    SALARY DEPARTMENT_ID
------------------------- ---------- -------------
Hartstein                     13000            20
Higgins                       12000           110
King                          26400            90
Hunold                         9000            60
```

Note: There is a correlation between outer query and inner query, because here in inner quey we are taking a reference of a column which exists in outer query's table. - Performancewise this query is poor, so that we can use inline view instead of correlated subquery.

7.6 Inline View

When we write a select statement in the FROM clause, that is known as inline view.

Example: same example of correlated subquery

-Find all employees who earn more than the average salary in their department

```
SQL> SELECT e.last_name, e.salary, e.department_id, b.avgsal FROM   employees e ,
  2  (SELECT department_id, AVG(salary)avgsal  FROM   employees  group by department_id) b
  3  WHERE e.department_id =b.department_id and e.salary>b.avgsal;

LAST_NAME                      SALARY DEPARTMENT_ID     AVGSAL
------------------------- ---------- ------------- ----------
Hartstein                       13000            20       9500
Higgins                         12000           110      10150
King                            26400            90 20133.3333
Hunold                           9000            60     5760.6
```

7.7 EXISTS Operator

• The EXISTS operator tests for existence of rows in the results set of the subquery.

Example:Find Employees Who Have At Least One Person Reporting to Them.

```
SQL> SELECT employee_id, last_name FROM   employees outer WHERE EXISTS
  2  ( SELECT 'X' FROM   employees WHERE manager_id = outer.employee_id);

EMPLOYEE_ID LAST_NAME
----------- -------------------------
        148 Cambrault
        102 De Haan
        147 Errazuriz
        121 Fripp
        108 Greenberg
```

EXAMPLE: Find All Departments That Do Not Have Any Employees

```
SQL>  SELECT department_id, department_name FROM departments d WHERE NOT EXISTS (SELECT 'X'
  2  FROM  employees WHERE department_id = d.department_id);

DEPARTMENT_ID DEPARTMENT_NAME
------------- ------------------------------
          120 Treasury
          130 Corporate Tax
          140 Control And Credit
          150 Shareholder Services
          160 Benefits
```

Subquery with insert statement:

Subqueries also can be used with INSERT statements. The INSERT statement uses the data returned from the subquery to insert into another table. The selected data in the subquery can be modified with any of the character, date or number functions.

The basic syntax is as follows:

INSERT INTO table_name [(column1 [, column2])]

SELECT [* | column1 [, column2]

FROM table1 [, table2]

[WHERE VALUE OPERATOR]

Example: Consider a table EMPLOYEES_BKP with similar structure as EMPLOYEESS table. Now to copy complete EMPLYEES table into EMPLOYEES_BKP, following is the syntax:

```
SQL> INSERT INTO EMPLOYEES_BKP
  2 SELECT * FROM EMPLOYEES WHERE EMPLOYEE_ID IN (SELECT EMPLOYEE_ID FROM CUSTOMERS) ;
```

Subqueries with the UPDATE Statement:

The subquery can be used in conjunction with the UPDATE statement. Either single or multiple columns in a table can be updated when using a subquery with the UPDATE statement.

The basic syntax is as follows:

```
UPDATE table
 SET column_name = new_value
 [ WHERE OPERATOR [ VALUE ]
(SELECT COLUMN_NAME FROM TABLE_NAME)
 [ WHERE) ]
```

EXAMPLE:

Assuming, we have EMPLOYEES_BKP table available which is backup of EMPLOYEES table.

Following example updates SALARY by 0.25 times in EMPLOYEES table for all the EMPLOYEES whose DEPARTMENT_ID is greater than or equal to 40?

```
UPDATE EMPLOYEES SET SALARY = SALARY * 0.25     WHERE DEPARTMENT_ID IN (SELECT DEPARTMENT_ID FROM
EMPLOYEES_BKP WHERE DEPARTMENT_ID >= 40 );
```

Subqueries with the DELETE Statement:

The subquery can be used in conjunction with the DELETE statement like with any other statements mentioned above.

The basic syntax is as follows:

```
DELETE FROM TABLE_NAME
 [ WHERE OPERATOR [ VALUE ]
(SELECT COLUMN_NAME FROM TABLE_NAME)
[ WHERE) ]
```

EXAMPLE:

Assuming, we have EMPLOYEES_BKP table available which is backup of EMPLOYEES table.

Following example deletes records from EMPLOYEES table for all the customers whose DEPARTMEN_ID IS equal to 50????

```
DELETE FROM EMPLOYEES WHERE DEPARTMENT_ID IN (SELECT DEPARTMENT_ID FROM CUSTOMERS_BKP WHERE
DEPARTMENT_ID = 50 );
```

Questions:

1. Create a report that displays the employee number, last name, and salary of all employees who earn more than the average salary. Sort the results in ascending order by salary.
2. Write a query that displays the employee no , last name of all employees who work in a department with any employee whose last name contains u.
3. The HR department needs a report that displays the last name, department number, and job ID of all employees whose department location ID is 1700.
4. displays the last name and salary of every employee who reports to King.
5. Write a query that displays the employee number and last name of all employees who work in a department with any employee whose last name contains a u.
6. Create a report that display the employee number, last name and salary of all employees who earn more than the average salary.
7. Display the last name , department number and job id of all employees whose location id is 1700.

8. Set Operators

Set operators combine the results of two or more component queries into one result. Queries containing set operators are called compound queries.

OPERATORS	RETURNS
UNION	All distinct rows selected by either query
UNION ALL	All rows selected by either query, including all duplicates
MINUS	All distinct rows that are selected by the first SELECT statement and not selected in the second SELECT statement
INTERSECT	All distinct rows selected by both queries

UNION Operator

The SQL UNION clause/operator is used to combine the results of two or more SELECT statements without returning any duplicate rows. To use UNION, each SELECT must have the same number of columns selected, the same number of column expressions, the same data type, and have them in the same order, but they do not have to be the same length

Example: Display the current and previous job details of all employees. Display each combination only once.

```
SQL>  SELECT  employee_id, job_id FROM employees
  2   UNION
  3   SELECT  employee_id, job_id FROM employees job_history;
```

UNION ALL Operator

The UNION operator returns results from both queries, including duplications.

Example:

Display the current and previous departments of all employees.

```
SQL> SELECT employee_id, job_id, department_id   FROM    employees
  2  UNION ALL
  3  SELECT employee_id, job_id, department_id   FROM    job_history   ORDER BY employee_id;
```

INTERSECT Operator

The SQL INTERSECT clause/operator is used to combine two SELECT statements, but returns rows only from the first SELECT statement that are identical to a row in the second SELECT statement. This means INTERSECT returns only common rows returned by the two SELECT statements. Just as with the UNION operator, the same rules apply when using the INTERSECT operator

Example: -Display the employee IDs and job IDs of those employees who currently have a job title that is the same as a previous jobtitle.

```
SQL>  SELECT employee_id, job_id    FROM    employees
  2  INTERSECT
  3  SELECT employee_id, job_id FROM job_history;

EMPLOYEE_ID JOB_ID
----------- ----------
        176 SA_REP
        200 AD_ASST
```

MINUS Operator

The MINUS operator returns rows in first query that are not present in the second query.

Example: - Display the employee IDs of those employees who have not changed their jobs even once.

```
SQL>  SELECT  employee_id FROM    employees
  2  minuS
  3  SELECT  employee_id  FROM    job_history;

EMPLOYEE_ID
-----------
        100
        103
        104
        105
```

Matching the SELECT Statements

The expressions in the select lists of the queries must match in number,you can use dummy columns and the data type conversion functions to comply with this rule.

```
SQL>  SELECT   employee_id, job_id,salary FROM employees
  2   UNION
  3  SELECT   employee_id, job_id,0 FROM job_history;

EMPLOYEE_ID JOB_ID         SALARY
----------- ---------- ----------
        100 AD_PRES         26400
        101 AC_ACCOUNT          0
        101 AC_MGR              0
        101 AD_VP           17000
        102 AD_VP           17000
```

9. MANIPULATING DATA

Data Manipulation Language

A DML statement is executed when you:

– Add new rows to a table

– Modify existing rows in a table

– Remove existing rows from a table

Types Of Data Manipulation Language(DML)

- Insert
- Update
- Delete
- Merge

INSERT Statement Syntax

Add new rows to a table by using the INSERT statement.

INSERT INTO table [(column [, column...])]

VALUES (value [, value...]);

-With this syntax, only one row is inserted at a time

Inserting New Rows

- Insert a new row containing values for each column.
- List values in the default order of the columns in the table.
- Optionally, list the columns in the INSERT clause.

Example:

```
SQL> INSERT INTO departments(department_id,department_name,
  2  manager_id, location_id) VALUES (370, 'Public Relations', 129, 1700);

1 row created.
```

Inserting Rows with Null Values

Implicit method: Omit the column from the column list.

```
SQL> INSERT INTO departments (department_id,department_name )
  2  VALUES (330, 'Purchasing');

1 row created.
```

Explicit method: Specify the NULL keyword in the VALUES clause.

```
SQL> INSERT INTO departments values (300,'finance',null,null );

1 row created.
```

Inserting Special Values

The SYSDATE function records the current date and time.

```
SQL> INSERT INTO EMP50 VALUES ('AHMED','a@GMAIL.COM',SYSDATE,300);

1 row created.
```

Creating a Script

- Use & substitution in a SQL statement to prompt for values.
- & is a placeholder for the variable value.

```
SQL> INSERT INTO departments (department_id, department_name, location_id)  VALUES
  2  (&department_id, '&department_name',&location);
Enter value for department_id: 340
Enter value for department_name: CIVIL
Enter value for location: 1700
old   2: (&department_id, '&department_name',&location)
new   2: (340, 'CIVIL',1700)

1 row created.
```

Note: This enables you to run the same script file over and over but supply a different set of values each time you run it.

Copying Rows from Another Table

Write your INSERT statement with a subquery.

Subqueries also can be used with INSERT statements. The INSERT statement uses the data returned from the subquery to insert into another table. The selected data in the subquery can be modified with any of the character, date or number functions.

The basic syntax is as follows:

INSERT INTO table_name [(column1 [, column2])]

SELECT [* | column1 [, column2]

FROM table1 [, table2]

[WHERE VALUE OPERATOR]

Example: Consider a table EMPLOYEES_BKP with similar structure as EMPLOYEESS table. Now to copy complete EMPLYEES table into EMPLOYEES_BKP, following is the syntax:

```
SQL> INSERT INTO EMPLOYEES_BKP
  2 SELECT * FROM EMPLOYEES WHERE EMPLOYEE_ID IN (SELECT EMPLOYEE_ID FROM CUSTOMERS) ;
```

Multitable INSERT Statements

The different types of multitable INSERT statements are:

- Unconditional INSERT
- Conditional ALL INSERT

Unconditional Insert:

It will insert the rows into all targeted tables.

Syntax:

```
INSERT ALL
INTO table_a VALUES(…,…,…)
INTO table_b VALUES(…,…,…)
INTO table_c VALUES(…,…,…)
SELECT … FROM source_table
WHERE …;
```

```
SQL> create table sal_his(empid number,hd date,sal number);

Table created.

SQL> create table mgr_his (empid number,mgrid number,sal number);

Table created.

SQL> insert all into sal_his values(employee_id,hire_date,salary)
  2  into mgr_his values(employee_id,manager_id,salary)
  3  select employee_id,hire_date,salary,manager_id from employees;

214 rows created.
```

Conditional ALL INSERT

- Specify the conditional_insert_clause to perform a conditional multitable INSERT.

- The Oracle server filters each insert_into_clause through the corresponding WHEN condition,which determines whether that insert_into_clause is executed.

- A single multitable INSERTstatement can contain up to 127 WHEN clauses.

```
SQL>  INSERT ALL
  2  WHEN SALARY > 10000 THEN  INTO sal_his VALUES(employee_id,hire_date,salary)
  3  WHEN MANAGER_ID > 200  THEN INTO mgr_his VALUES(employee_id,manager_id,salary)
  4  select employee_id,hire_date,salary,manager_id from employees;

17 rows created.
```

UPDATE Statement Syntax

The SQL UPDATE Query is used to modify the existing records in a table. You can use WHERE clause with UPDATE query to update selected rows, otherwise all the rows would be affected.

Syntax:

UPDATE table_name SET column1 = value1, column2 = value2 ...,

columnN = valueN

WHERE [condition];

```
SQL> update emp set salary=100000 where employee_id=100;

4 rows updated.

SQL> select employee_id,salary from emp where employee_id=100;

EMPLOYEE_ID     SALARY
----------- ----------
        100     100000
        100     100000
        100     100000
        100     100000
```

NOTE: If you forgot to mention the Where clause then salary of all the employees will het updated.

Updating Two Columns with a Subquery

Example: Update employee 114's job and salary to match that of employee 205.?

```
SQL> UPDATE  employees SET job_id =
  2  (SELECT job_id  FROM employees  WHERE  employee_id = 205),
  3  salary = (SELECT salary FROM   employees  WHERE  employee_id = 205)
  4  WHERE employee_id    = 114;
```

Updating Rows Based on Another Table

Use subqueries in UPDATE statements to update rows in a table based on values from another table.

```
SQL>  UPDATE emp  SET department_id=(SELECT department_id  FROM employees WHERE employee_id =100)
  2  WHERE job_id  = (SELECT job_id FROM employees WHERE employee_id = 200);

4 rows updated.
```

Correlated UPDATE

Use a correlated subquery to update rows in one table based on rows from another table.

Syntax:

UPDATE table1 alias1 SET column = (SELECT expressio FROM table2 alias2

WHERE alias1.column = alias2.column);

Example:

Assuming, we have EMPLOYEES_BKP table available which is backup of EMPLOYEES table.

Following example updates SALARY by 0.25 times in EMPLOYEES table for all the EMPLOYEES whose DEPARTMENT_ID is greater than or equal to 40?

UPDATE EMPLOYEES SET SALARY = SALARY * 0.25 WHERE DEPARTMENT_ID IN (SELECT DEPARTMENT_ID FROM EMPLOYEES_BKP WHERE DEPARTMENT_ID >= 40);

DELETE Statement

The SQL DELETE Query is used to delete the existing records from a table. You can use WHERE clause with DELETE query to delete selected rows, otherwise all the records would be deleted.

Syntax:

DELETE FROM table_name

WHERE [condition];

```
SQL>  DELETE FROM departments   WHERE department_name = 'Payroll';

1 row deleted.
```

NOTE: All rows in the table are deleted if you omit the WHERE clause.

Deleting Rows Based on Another Table

Use subqueries in DELETE statements to remove rows from a table based on values from another table.

```
SQL> DELETE FROM employees   WHERE department_id =
  2  (SELECT department_id FROM departments WHERE department_name='Shipping');
```

Correlated DELETE

Use a correlated subquery to delete rows in one table based on rows from another table.

The subquery can be used in conjunction with the DELETE statement like with any other statements mentioned above.

The basic **syntax** is as follows:

DELETE FROM TABLE_NAME

[WHERE OPERATOR [VALUE]

(SELECT COLUMN_NAME FROM TABLE_NAME)

[WHERE)]

EXAMPLE:

Assuming, we have EMPLOYEES_BKP table available which is backup of EMPLOYEES table.

Following example deletes records from EMPLOYEES table for all the customers whose DEPARTMEN_ID IS equal to 50????

DELETE FROM EMPLOYEES WHERE DEPARTMENT_ID IN (SELECT DEPARTMENT_ID FROM CUSTOMERS_BKP WHERE DEPARTMENT_ID = 50);

MERGE Statement

Use the **MERGE** statement to select rows from one or more sources for update or insertion into a table or view. You can specify conditions to determine whether to update or insert into the target table or view.

This statement is a convenient way to combine multiple operations. It lets you avoid multiple **INSERT**, **UPDATE**, and **DELETE** DML statements.

MERGE is a deterministic statement. You cannot update the same row of the target table multiple times in the same **MERGE** statement.

• Provides the ability to conditionally update or insert data into a database table

• Performs an UPDATE if the row exists, and an INSERT if it is a new row: – Avoids separate updates – Increases performance and ease of use

Syntax:

MERGE INTO table_name table_alias

USING (table|view|sub_query) alias

ON (join condition)

WHEN MATCHED

THEN UPDATE SET col1 = col_val1,col2 = col2_val

WHEN MATCHED

THEN INSERT (column_list) VALUES (column_values);

Suppose we are having two tables master and slave.which having certain data in it.like.

```
SQL> select * from master;

        ID NAME                 PLAN
---------- -------------------- --------------------
        10 Rash                 Prepaid
        20 Kajal                Postpaid
        30 Pinki                Prepaid

SQL> select * from slave;

        ID NAME                 PLAN
---------- -------------------- --------------------
        10 Rash                 Postpaid
        30 Pinki                Postpaid
        40 Riaz                 Prepaid
        50 Jai                  Prepaid
```

Now we are trying to merge these two table on certa in condition like update when id column of master table is same as id column of slave other wise insert new row in master.

```
SQL> merge into master m using slave s on (m.id=s.id)
  2  when matched then update set m.name = s.name
  3  ,m.plan=s.plan when not matched then insert values (s.id,s.name,s.plan);

4 rows merged.
```

After running above query master table will get updated and have new values.

```
SQL> select * from master;

        ID NAME                 PLAN
---------- -------------------- --------------------
        10 Rash                 Postpaid
        20 Kajal                Postpaid
        30 Pinki                Postpaid
        50 Jai                  Prepaid
        40 Riaz                 Prepaid
```

TRUNCATE Statement

- Removes all rows from a table, leaving the table empty and the table structure remains.
- Is a data definition language (DDL) statement rather than a DML statement.

Syntax: TRUNCATE TABLE table_name;

TCL (Transaction Control Language) statements

-**Commit**

-**Rollback**

-**Savepoint**

Database Transactions

• Begin when the first DML SQL statement is executed

• End with one of the following events:

– A COMMIT or ROLLBACK statement is issued.

– A DDL or DCL statement executes (automatic commit).

– The user exits SQL Developer or SQL*Plus. – The system crashes.

Advantages of COMMIT and ROLLBACK Statements

With COMMIT and ROLLBACK statements, you can:

- Ensure data consistency
- Preview data changes before making changes permanent

Savepoint Statement

• Create a marker in a current transaction by using the SAVEPOINT statement.

• Roll back to that marker by using the ROLLBACK TO SAVEPOINT statement.

Example: update employees set salary=5000 where employee_id=101;

savepoint a;

update employees set salary=10000 where job_id = 'IT_PROG';

savepoint b;

rollback to a;

Implicit Transaction Processing

• An automatic commit occurs under the following circumstances:

– DDL statement is issued

– DCL statement is issued (grant,revoke)

– Normal exit from SQL Developer or SQL*Plus, without

explicitly issuing COMMIT or ROLLBACK statements

• An automatic rollback occurs under an abnormal termination of SQL Developer or SQL*Plus or a system failure.

Committing Data

• Commit the changes permanent:

Commit;

10. DDL Statement To Create or merge Table

CREATE TABLE Statement

You must have:

– The CREATE TABLE privilege

– A storage area

SYNTAX:

CREATE TABLE [schema.]table

(column datatype [DEFAULT expr][, ...]);

- You specify the:

– Table name

– Column name, column data type, and column size Referencing Another User's Tables

- Tables belonging to other users are not in the user's schema.

- You should use the owner's name as a prefix to those tables.

Example: suppose we want to access a employees table of userb then we use following syntax.

Select * from userb.employees

DEFAULT Option

Specify a default value for a column during an insert.

Example:

```
SQL> create table emp (last_name varchar2(30), hire_date date default sysdate);

Table created.
```

DATATYPE IN SQL

CHARACTER DATA TYPES IN SQL

Data Type Syntax	Oracle 9i	Oracle 10g	Oracle 11g	Explanation
char(size)	Maximum size of 2000 bytes.	Maximum size of 2000 bytes.	Maximum size of 2000 bytes.	Where ***size*** is the number of characters to store. Fixed-length strings. Space padded.
nchar(size)	Maximum size of 2000 bytes.	Maximum size of 2000 bytes.	Maximum size of 2000 bytes.	Where ***size*** is the number of characters to store. Fixed-length NLS string Space padded.
nvarchar2(size)	Maximum size of 4000 bytes.	Maximum size of 4000 bytes.	Maximum size of 4000 bytes.	Where ***size*** is the number of characters to store. Variable-length NLS string.
varchar2(size)	Maximum size of 4000 bytes. Maximum size of 32KB in PLSQL.	Maximum size of 4000 bytes. Maximum size of 32KB in PLSQL.	Maximum size of 4000 bytes. Maximum size of 32KB in PLSQL.	Where ***size*** is the number of characters to store. Variable-length string.
long	Maximum size of 2GB.	Maximum size of 2GB.	Maximum size of 2GB.	Variable-length strings. (backward compatible)
raw	Maximum size of 2000 bytes.	Maximum size of 2000 bytes.	Maximum size of 2000 bytes.	Variable-length binary strings
long raw	Maximum size of 2GB.	Maximum size of 2GB.	Maximum size of 2GB.	Variable-length binary strings. (backward compatible)

Number Data Type in SQL

Data Type Syntax	Oracle 9i	Oracle 10g	Oracle 11g	Explanation
number(p,s)	Precision can range from 1 to 38. Scale can range from -84 to 127.	Precision can range from 1 to 38. Scale can range from -84 to 127.	Precision can range from 1 to 38. Scale can range from -84 to 127.	Where ***p*** is the precision and ***s*** is the scale. For example, number(7,2) is a number that has 5 digits before the decimal and 2 digits after the decimal.
numeric(p,s)	Precision can range from 1 to 38.	Precision can range from 1 to 38.	Precision can range from 1 to 38.	Where ***p*** is the precision and ***s*** is the scale. For example, numeric(7,2) is a number that has 5 digits before the decimal and 2 digits after the decimal.
float				
dec(p,s)	Precision can range from 1 to 38.	Precision can range from 1 to 38.	Precision can range from 1 to 38.	Where ***p*** is the precision and ***s*** is the scale. For example, dec(3,1) is a number that has 2 digits before the decimal and 1 digit after the decimal.
decimal(p,s)	Precision can range from 1 to 38.	Precision can range from 1 to 38.	Precision can range from 1 to 38.	Where ***p*** is the precision and ***s*** is the scale. For example, decimal(3,1) is a number that has 2 digits before the decimal and 1 digit after the decimal.

Date / Time datatype in sql

Data Type Syntax	Oracle 9i	Oracle 10g	Oracle 11g	Explanation
date	A date between Jan 1, 4712 BC and Dec 31, 9999 AD.	A date between Jan 1, 4712 BC and Dec 31, 9999 AD.	A date between Jan 1, 4712 BC and Dec 31, 9999 AD.	
timestamp (*fractional seconds precision*)	***fractional seconds precision*** must be a number between 0 and 9. (default is 6)	***fractional seconds precision*** must be a number between 0 and 9. (default is 6)	***fractional seconds precision*** must be a number between 0 and 9. (default is 6)	Includes year, month, day, hour, minute, and seconds. For example: timestamp(6)
timestamp (*fractional seconds precision*) with time zone	***fractional seconds precision*** must be a number between 0 and 9. (default is 6)	***fractional seconds precision*** must be a number between 0 and 9. (default is 6)	***fractional seconds precision*** must be a number between 0 and 9. (default is 6)	Includes year, month, day, hour, minute, and seconds; with a time zone displacement value. For example: timestamp(5) with time zone
timestamp (*fractional seconds precision*) with local time zone	***fractional seconds precision*** must be a number between 0 and 9. (default is 6)	***fractional seconds precision*** must be a number between 0 and 9. (default is 6)	***fractional seconds precision*** must be a number between 0 and 9. (default is 6)	Includes year, month, day, hour, minute, and seconds; with a time zone expressed as the session time zone. For example: timestamp(4) with local time zone
interval year (*year precision*) to month	***year precision*** is the number of digits in the year. (default is 2)	***year precision*** is the number of digits in the year. (default is 2)	***year precision*** is the number of digits in the year. (default is 2)	Time period stored in years and months. For example: interval year(4) to month

Large object Datatypes

Data Type Syntax	Oracle 9i	Oracle 10g	Oracle 11g	Explanation
bfile	Maximum file size of 4GB.	Maximum file size of 2^{32}-1 bytes.	Maximum file size of 2^{64}-1 bytes.	File locators that point to a binary file on the server file system (outside the database).
blob	Store up to 4GB of binary data.	Store up to (4 gigabytes -1) * (the value of the CHUNK parameter of LOB storage).	Store up to (4 gigabytes -1) * (the value of the CHUNK parameter of LOB storage).	Stores unstructured binary large objects.
clob	Store up to 4GB of character data.	Store up to (4 gigabytes -1) * (the value of the CHUNK parameter of LOB storage) of character data.	Store up to (4 gigabytes -1) * (the value of the CHUNK parameter of LOB storage) of character data.	Stores single-byte and multi-byte character data.
nclob	Store up to 4GB of character text data.	Store up to (4 gigabytes -1) * (the value of the CHUNK parameter of LOB storage) of character text data.	Store up to (4 gigabytes -1) * (the value of the CHUNK parameter of LOB storage) of character text data.	Stores unicode data.

Creating a Table by Using a Subquery

```
SQL> CREATE TABLE dept80 AS
  2  SELECT employee_id, last_name,salary*12 ANNSAL,hire_date FROM employees
  3  where department_id=80;

Table created.
```

Note: when you create table from subquery only not null constraint will apply new table.

Including Constraints

Constraint is restriction on a table which allows only valid enteries into table's column.

• Constraints enforce rules at the table level.

• Constraints prevent the deletion of a table if there are dependencies.

• The following constraint types are valid:

- **NOT NULL**
- **UNIQUE**
- **PRIMARY KEY**
- **FOREIGN KEY**
- **CHECK**

Constraint	Explanation
NOT NULL	Specifies that the column cannot contain a null value
UNIQUE	Specifies a column or combination of columns whose values must be unique for all rows in the table
PRIMARY KEY	Uniquely identifies each row of the table
FOREIGN KEY	Establishes and enforces a foreign key relationship Between the column and a column of the referenced
CHECK	Specifies a condition that must be true

Constraint Guidelines

• You can name a constraint, or the Oracle server generates a name with the SYS_Cn format.

• Create a constraint at either of the following times:

– At the same time as the table is created.

– After the table has been created

• Define a constraint at the column or table level.

• View a constraint in the data dictionary.

Defining Constraints

Column-level constraint

When we mentioned constraint at the time of declararing column the we call it column level constraint.

```
SQL>  CREATE TABLE emp1
  2  (employee_id NUMBER(6) CONSTRAINT emp_id_pk PRIMARY KEY,first_name VARCHAR2(20));

Table created.
```

Table-level constraint

When we mentioned all constraint at the after declaring all column then we call it table level constraint.

```
SQL>  CREATE TABLE emp2(employee_id NUMBER(6),first_name VARCHAR2(20),
  2  constraint emp2_id_pk primary key(employee_id));

Table created.
```

Constraints:

NOT NULL Constraint

Ensures that null values are not permitted for the column.

You cannot apply not null constraint on table level.

Example:

```
SQL> create table emp3(id number not null);

Table created.
```

If you forget to apply not null constraint on column at the time of creation the you can use alter command.

```
SQL> Alter table emp3 modify id not null;
Alter table emp3 modify id not null
```

UNIQUE Constraint

--No two rows of a table can have duplicate values in a specified column or set of columns.

--If the UNIQUE constraint comprises more than one column, that group of columns is called a composite unique key.

--Null values are allowed in the UNIQUE constraint.

--Can be defined at either table level or column level.

-- As we define the unique constraint unique index is automatically created.

Example:(column-level)

create table emp(emp_id number, email varchar2(30)unique);

Example:(table-level)

create table emp(emp_id number, email varchar2(30),e_name varchar2(22), constraint emp_email_uk unique(email);

PRIMARY KEY Constraint

--Is the combination of the UNIQUE and NOT NULL constraint.

--Only one primary key can be created for each table.

-- unique index is automatically created.

Example:(column-level)

create table emp(id number primary_key);

Example:(table-level)

create table emp(id number,name varchar2(20),email varchar2(30) constraint emp_id_pk primary key(id));

FOREIGN KEY Constraint:

--Restricts invalid values to be inserted. The valid values are referenced in a pk or uk column in the same or different table.

- multiple fk constraints can be applied on a table

- can be created on composite columns (on table level)

- can be added in both column/table level

- 1 / multiple columns can refer to a single PK.

- 1 column can't refer to multiple PK.

- null values can be present

Foreign Key Rules

- delete rule (on parent table)

- by default we can't delete a row in the parent table, which is referenced by child records.

- **on delete cascade**: child records will be deleted if parent record is deleted.

- **on delete set null**: child record values will be set to null value, if parent record is deleted.

- **insert/update rule (on child table)**

- no values can be inserted or updated if the value is not present in the parent table (referenced table).

Example: :(column-level)

Create table emp (eid number,did number references dept(did));

Example:(table-level)

CREATE TABLE employees(employee_id NUMBER(6), last_name VARCHAR2(25), department_id number(4), constraint emp_dept_fk foreign key (department_id) references departments(department_id)[ON DELETE CASCADE | ON DELETE SET NULL]);

CHECK Constraint

- Defines a condition that each row must satisfy
- The following expressions are not allowed:

– References to CURRVAL, NEXTVAL, LEVEL, and ROWNUM pseudocolumns

– Calls to SYSDATE, USER, functions – Queries that refer to other values in other rows

Example:

create table employee(emp_id number,salary check (salary>1000));

Enabling and Disabling a constraint:

Syntax:

ALTER TABLE table_name disable constraint constraint_name [CASCADE];

ALTER TABLE table_name enable constraint constraint_name;

ALTER TABLE Statement

Use the ALTER TABLE statement to:

- Add a constraint in an existing table
- Add a new column
- Modify an existing column
- Define a default value for the new column
- Drop a column

Example: Adding a constraint

ALTER table employee add constraint emp_id_pk primary key(employee_id);

Alter table employee modify employee_id not null;

Example: Adding a new column

ALTER table employees add sal number(3);

EXAMPLE: Modify an existing column

ALTER table employees modify sal number(8,2);

EXAMPLE: Define a default value

ALTER table employees modify DOB default sysdate;

EXAMPLE : Dropping a column

ALTER table employees drop column job_id;

Dropping a Table

- All data and structure in the table are deleted.
- Any pending transactions are committed.
- All indexes are dropped.
- All constraints are dropped.
- You cannot roll back the DROP TABLE statement.

Syntax:

DROP TABLE table_name

Questions:

1. Create a table named EMP, with column emp_no,ename, sal,dep_no,hire_date.
2. Create a table named DEPT, with column dep_no,d_name,city,loc_id.
3. Add a constraint in EMP table :

   a.Primary key on emp_no

   b.Foreign key on dep_no (with cascade option) referenced from the DEPT table.

   c.Check salary should be greater than 5000 and name should be in upper case.

11. Creating Other Schema Objects

Objectives

After completing this lesson, you should be able to do the following:

- Create simple and complex views
- Retrieve data from views
- Create, maintain, and use sequences
- Create and maintain indexes
- Create private and public synonyms

Database Objects:

-**Table**

-**View**

-**Sequence**

- **index**

-**synonym**

Table :

Is the basic unit of storage, and consist of rows and columns.

What Is a View?

A view is a virtual table.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

Advantages of Views

-- To restrict data access

-- To make complex queries easy

-- To provide data independence

Types of VIEWS

-Simple Views

-Complex Views

A simple view is one that:

- Derives data from only one table

- Contains no functions or groups of data

- Can perform DML operations through the view

A complex view is one that:

- Derives data from manytables

- Contains functions or groups of data

- Does not always allow DML operations through the view

Creating a View Syntax:

CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW view _name[(alias[, alias]...)]

AS subquery [WITH CHECK OPTION [CONSTRAINT constraint]]

[WITH READ ONLY [CONSTRAINT constraint]];

EXAMPLE:

Create the EMPVU80 view, which contains details of employees in department 80.?

```
SQL> CREATE VIEW empvu80 AS SELECT  employee_id, last_name,
  2  salary FROM   employees WHERE  department_id = 80;

View created.
```

Example: Create a view by using column aliases in the subquery?

```
SQL> CREATE VIEW salvu50 AS SELECT employee_id ID_NUMBER, last_name NAME,
  2  salary*12 ANN_SALARY FROM  employees WHERE department_id = 50;

View created.
```

Retrieving Data from a View

```
SQL> SELECT * FROM   salvu50;

 ID_NUMBER NAME                      ANN_SALARY
---------- ------------------------- ----------
       198 OConnell                       31200
       199 Grant                          31200
       120 Weiss                          96000
```

Modifying a View

- Modify the EMPVU80 view by using a CREATE OR REPLACE VIEW clause. Add an alias for each column name:

```
SQL>  CREATE OR REPLACE VIEW empvu80 (id_number, name, sal, department_id)
  2  AS SELECT employee_id, first_name || ' '              || last_name, salary, department_id
  3  FROM employees   WHERE department_id = 80;

View created.
```

NOTE: Column aliases in the CREATE OR REPLACE view clause are listed in the same order as the columns in the same query.

Creating a Complex View

Create a complex view that contains group functions to display values from two tables.

```
SQL> CREATE OR REPLACE VIEW dept_sum_vu (name, minsal, maxsal)
  2   AS SELECT d.department_name, MIN(e.salary),MAX(e.salary) FROM employees e
  3  JOIN departments d on e.department_id=d.department_id group by d.department_name;

View created.
```

Rules for Performing DML Operations on a View

• You can usually perform DML operations on simple views.

• You cannot remove a row if the view contains the following:

– Group functions

– A GROUP BY clause

– The DISTINCT keyword

– The pseudocolumn ROWNUM keyword

You cannot modify (UPDATE)data in a view if it contains:

• Group functions • A GROUP BY clause

• The DISTINCT keyword

• The pseudocolumn ROWNUM keyword

• Columns defined by expressions

You cannot add data through a view if the view includes:

• Group functions

• A GROUP BY clause

• The DISTINCT keyword

• The pseudocolumn ROWNUM keyword

• Columns defined by expressions

• NOT NULL columns in the base tables that are not selected by the view

Using the WITH CHECK OPTION Clause

• You can ensure that DML operations performed on the view stay in the domain of the view by using the WITH CHECK OPTION clause:

Example:

```
SQL>  CREATE OR REPLACE VIEW empvu20 AS SELECT *
  2  FROM  employees   WHERE    department_id = 20   WITH CHECK OPTION CONSTRAINT empvu20_ck;

View created.
```

• Any attempt to change the department number for any row in the view fails because it violates the WITH CHECK OPTION constraint.

Denying DML Operations

• You can ensure that no DML operations occur by adding the WITH READ ONLY option to your view definition.

```
SQL> CREATE OR REPLACE VIEW empvu10 (employee_number, employee_name) AS SELECT
  2   employee_id, last_name  FROM  employees WHERE    department_id = 10   WITH READ ONLY;

View created.
```

• Any attempt to perform a DML operation on any row in the view results in an Oracle server error.

Removing a View

You can remove a view without losing data because a view is based on underlying tables in the database. Syntax: DROP VIEW view_name;

```
SQL> drop view empvu10;

View dropped.
```

Sequences

Use the CREATE SEQUENCE statement to create a **sequence**, which is a database object from which multiple users may generate unique integers. You can use sequences to automatically generate primary key values.

When a sequence number is generated, the sequence is incremented, independent of the transaction committing or rolling back. If two users concurrently increment the same sequence, then the sequence numbers each user acquires may have gaps, because sequence numbers are being generated by the other user. One user can never acquire the sequence number generated by another user. Once a sequence value is generated by one user, that user can continue to access that value regardless of whether the sequence is incremented by another user.

Sequence numbers are generated independently of tables, so the same sequence can be used for one or for multiple tables. It is possible that individual sequence numbers will appear to be skipped, because they were generated and used in a transaction that ultimately rolled back. Additionally, a single user may not realize that other users are drawing from the same sequence.

Once a sequence is created, you can access its values in SQL statements with the CURRVAL pseudocolumn, which returns the current value of the sequence, or the NEXTVAL pseudocolumn, which increments the sequence and returns the new value.

Advantages of sequence:

- Can automatically generate unique numbers
- Is a sharable object
- Can be used to create a primary key value

Syntax:

CREATE SEQUENCE sequence

[INCREMENT BY n]

[START WITH n]

[{MAXVALUE n|NOMAXVALUE}]

[{MINVALUE n | NOMINVALUE}]

[{CYCLE | NOCYCLE }]

[{CACHE n | NOCACHE }]

Creating a Sequence Example:

- Create a sequence named DEPT_DEPTID_SEQ to be used for the primary key of the DEPARTMENTS table.

```
SQL>  CREATE sequence dept_id_seq increment by 10 start with 120 maxvalue 9999 nocache nocycle;

Sequence created.
```

NEXTVAL and CURRVAL Pseudocolumns

- NEXTVAL returns the next available sequence value. It returns a unique value every time it is referenced, even for different users.
- CURRVAL obtains the current sequence value.

Using a Sequence

- Insert a new department named “Support” in location ID 2500?

INSERT INTO departments(department_id,department_name,location_id) VALUES (dept_id_seq.NEXTVAL, 'Support', 2500);

- View the current value for the DEPT_ID_SEQ sequence.

```
SQL> select dept_id_seq.currval from dual;

   CURRVAL
----------
       120
```

Caching Sequence Values

• Caching sequence values in memory gives faster access to those values.

• Gaps in sequence values can occur when:

– A rollback occurs

– The system crashes

– A sequence is used in another table

Modifying a Sequence

Change the increment value, maximum value, minimum value, cycle option, or cache option.

Example:

```
SQL> ALTER SEQUENCE dept_id_seq
  2  INCREMENT BY 20
  3  MAXVALUE 999999
  4  NOCYCLE
  5  NOCACHE;

Sequence altered.
```

Guidelines for Modifying a Sequence

• You must be the owner or have the ALTER privilege for the sequence.

• Only future sequence numbers are affected.

• The sequence must be dropped and re-created to restart the sequence at a different number.

• Some validation is performed.

DROPPING a sequence

Syntax:

DROP sequence sequence_name;

```
SQL> DROP sequence dept_id_seq;

Sequence dropped.
```

Indexes

• Is a schema object

• Can be used by the Oracle server to speed up the retrieval of rows by using a pointer

• Can reduce disk I/O by using a rapid path access method to locate data quickly

• Is used and maintained automatically by the Oracle server

How Are Indexes Created?

• **Automatically**: A unique index is created automatically when you define a PRIMARY KEY or UNIQUE constraint in a table definition.

• **Manually:** Users can create nonunique indexes on columns to speed up access to the rows.

Creating an Index

Create an index on one or more columns.

Syntax:

CREATE INDEX index_name ON table (column[, column]...);

Example:

Improve the speed of query access to the LAST_NAME column in the EMPLOYEES table.?

```
SQL> create index emp_lname_idx on employees(last_name);

Index created.
```

Create an index when:

- A column contains a wide range of values

-A column contains a large number of null values

-One or more columns are frequently used together in a WHERE clause or a join condition

-The table is large and most queries are expected to retrieve less than 2% to 4% of the rows in the table

Do not create an index when

The columns are not often used as a condition in the query.

The table is small.

The table is updated frequently.

Removing an Index

• Remove an index from the data dictionary by using the DROP INDEX command.

Syntax:

DROP INDEX index_name;

To drop an index, you must be the owner of the index or have the DROP ANY INDEX privilege.

Synonyms

Simplify access to objects by creating a synonym (another name for an object). With synonyms, you can:

• Create an easier reference to a table that is owned by another user

• Shorten lengthy object names

Syntax:

CREATE [PUBLIC] SYNONYM synonym_name FOR object;

Example:

Create a shortened name for the DEPT_SUM_VU view?

```
SQL>  CREATE SYNONYM d_sum  FOR dept_sum_vu;

Synonym created.
```

Drop a synonym

Syntax:

drop synonym synonym name;

```
SQL> drop synonym d_sum;

Synonym dropped.
```

12. Managing Objects with Data Dictionary Views

Objectives

• Use the data dictionary views to research data on your objects

• Query various data dictionary views

Data Dictionary Structure Consists of

--Base tables

--User accessible views

View naming convention

View Prefix	Purpose
USER	User's view (what is in your schema; what you own)
ALL	What you can access
DBA	DBA's view (what is in everyone's schema) V$ Performance-related data
V$	Performance-related data

How to Use the Dictionary Views

Start with DICTIONARY. It contains the names and descriptions of the dictionary tables and views.

DESC DICTIONARY;

Example:

select * from dictionary where table_name='USER_OBJECTS';

Example: The following query returns the names of all views that you are permitted to access in which the COMMENTS column contains the word columns.

```
SQL>  SELECT table_name FROM dictionary WHERE LOWER(comments) LIKE '%columns';

TABLE_NAME
------------------------------
ALL_ENCRYPTED_COLUMNS
ALL_STREAMS_COLUMNS
ALL_UPDATABLE_COLUMNS
USER_CLU_COLUMNS
```

Note: The names in the data dictionary are uppercase.

USER_OBJECTS and ALL_OBJECTS Views

• Use the USER_OBJECTS view to:

– See all of the objects that are owned by you

– Obtain a listing of all object names and types in your schema, plus the following information:

- Date created

- Date of last modification

-Status (valid or invalid)

• Use the All_OBJECTS view to see all objects to which You have access.

USER_OBJECTS View

```
SQL> SELECT object_name, object_type, created, status FROM user_objects ORDER BY object_type;
```

Table Information (**USER_TABLES):**

DESC USER_TABLES;

NOTE: You can use the USER_TABLES view to obtain the names of all of your tables.

Column Information

USER_TAB_COLUMNS

DESC USER_TAB_COLUMNS;

This view contains information such as:

• Column names

• Column data types

• Length of data types

• Precision and scale for NUMBER columns

• Whether nulls are allowed (Is there a NOT NULL constraint on the column?)

• Default value

Example:

```
SQL> SELECT column_name, data_type, data_length,
  2  data_precision, data_scale, nullable FROM
  3  user_tab_columns WHERE table_name = 'EMPLOYEES';
```

Constraint Information

• USER_CONSTRAINTS describes the constraint definitions on your tables.

• USER_CONS_COLUMNS describes columns that are owned by you and that are specified in constraints.

DESC USER_CONS_COLUMNS;

Example:

```
SQL> SELECT constraint_name, constraint_type,search_condition, r_constraint_name,
  2   delete_rule, status FROM  user_constraints WHERE table_name = 'EMPLOYEES';
```

```
SQL>  SELECT constraint_name, column_name FROM user_cons_columns WHERE table_name='EMPLOYEES';
```

View Information

DESC USER_VIEWS;

```
SQL> SELECT DISTINCT view_name FROM user_views;

VIEW_NAME
------------------------------
EMP_DETAILS_VIEW
EMPVU20
EMP50
DEPT_SUM_VU
EMPVU80
```

```
SQL> SELECT text FROM user_views WHERE view_name = 'EMP_DETAILS_VIEW';

TEXT
--------------------------------------------------------------------------------
SELECT
  e.employee_id,
  e.job_id,
  e.manager_id,
  e.department_id,
  d.locat
```

Sequence Information

DESC USER_SEQUENCES;

```
SQL> SELECT sequence_name, min_value, max_value,increment_by,last_number FROM user_sequences;

SEQUENCE_NAME                  MIN_VALUE  MAX_VALUE INCREMENT_BY LAST_NUMBER
------------------------------ ---------- ---------- ------------ -----------
DEPARTMENTS_SEQ                         1       9990           10         280
EMPLOYEES_SEQ                           1 1.0000E+27            1         207
LOCATIONS_SEQ                           1       9900          100        3300
```

Note: The last_number column displays the next available sequence number if no cache is specified.

Synonym Information

DESC USER_SYNONYMS;

```
SQL> SELECT SYNONYM_NAME,TABLE_NAME FROM USER_SYNONYMS;

SYNONYM_NAME                   TABLE_NAME
------------------------------ ------------------------------
S2                             EMP
```

Adding Comments to a Table

- You can add comments to a table or column by using the COMMENT statement.

Syntax:

COMMENT ON TABLE table | COLUMN table.column IS 'text';

Example:

```
SQL> COMMENT ON TABLE employees  IS 'Employee Information';

Comment created.
```

- Comments can be viewed through the data dictionary views:

--ALL_COL_COMMENTS

--USER_COL_COMMENTS

--ALL_TAB_COMMENTS

--USER_TAB_COMMENTS

13. Controlling the User Access

Objectives

- Differentiate system privileges from object privileges
- Grant privileges on tables
- View privileges in the data dictionary
- Grant roles
- Distinguish between privileges and roles

Privileges

Privileges are right to execute particular SQL statements.

- System privileges: Gaining access to the database
- Object privileges: Manipulating the content of the database objects
- Schemas: Collection of objects such as tables, views, and sequences

System Privileges

- More than 100 privileges are available.
- The database administrator has high-level system privileges for tasks such as:

– Creating new users

– Removing users

– Removing tables

– Backing up tablesExample

Creating Users

The DBA creates users with the CREATE USER statement.

Syntax:

CREATE USER user_name IDENTIFIED BY password;

User System Privileges

- After a user is created, the DBA can grant specific system privileges to that user.

Syntax:

GRANT privilege [, privilege...]

TO user [, user| role, PUBLIC...];

- An application developer, for example, may have the following system privileges:

-- CREATE SESSION

-- CREATE TABLE

-- CREATE SEQUENCE

-- CREATE VIEW

Note: Current system privileges can be found in the SESSION_PRIVS dictionary view.

Granting System Privileges

The DBA can grant specific system privileges to a user.

GRANT create session, create table, create sequence, create view TO scott;

What Is a Role?

A role is a named group of related privileges that can be granted to the user. This method makes it easier to revoke and maintain privileges.

How to create role?

Syntax:

CREATE ROLE ROLE_NAME;

```
SQL> CREATE ROLE ASSISTANT;

Role created.
```

Grant privileges to a role?

```
SQL> GRANT create table, create view TO ASSISTANT;
```

GRANT a role to user:

Syntax:

GRANT ROLE to USER [User2,user3...];

EXAMPLE:

```
grant manager to bell,kochhar;
```

Changing Your Password

- The DBA creates your user account and initializes your password.
- You can change your password by using the ALTER USER statement.

Example:

```
ALTER USER HR IDENTIFIED BY employee;
```

Object Privileges

- Object privileges vary from object to object.
- An owner has all the privileges on the object.
- An owner can give specific privileges on that owner's object.

Syntax:

GRANT object_priv[(columns)]

ON object

to {user|role|public}

[with grant option];

Note:If the grant includes WITH GRANT OPTION, then the grantee can further grant the object privilege to other users; otherwise, the grantee can use the privilege but cannot grant it to other users.

Example:

GRANT select ON employees TO sue, rich;

- Grant privileges to update specific columns to users and roles:

Example:

```
GRANT update (department_name, location_id) ON departments TO scott, manager;
```

Revoking Object Privileges

- You use the REVOKE statement to revoke privileges granted to other users.

Syntax:

REVOKE {privilege [, privilege...]|ALL}

ON object

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

[CASCADE CONSTRAINTS];

Example: A DBA revoke the SELECT and INSERT privileges given to user Scott on the DEPARTMENTS table.

```
REVOKE select, insert ON  departments FROM   scott;
```

14. Flashback

Flashback table statement:

Repair tool for accidental table modifications

– Restores a table to an earlier point in time

– Benefits: Ease of use, availability, fast execution

– Performed in place

Syntax:

FLASHBACK TABLE[schema.]table[,[schema.]table]...

To { Timestamp | SCN } expr
[{ENABLE | DISABLE } TRIGGERS]

FLASHBACK TABLE Statement

Step 1 drop table.

```
SQL> drop table emp;

Table dropped.
```

Step 2 select from recyclebin

```
SQL> SELECT original_name, operation, droptime FROM recyclebin;

ORIGINAL_NAME                    OPERATION DROPTIME
-------------------------------- --------- -------------------
EMP                              DROP      2015-07-10:21:29:31
MGR_HIS                          DROP      2015-07-09:10:37:31
SAL_HI                           DROP      2015-07-09:10:37:21
```

Step 3 Now flashback it from recycle bin and select it from recyclebin

```
SQL> FLASHBACK table emp to before drop;

Flashback complete.

SQL> SELECT original_name, operation, droptime FROM recyclebin;

ORIGINAL_NAME                    OPERATION DROPTIME
-------------------------------- --------- -------------------
SAL_HI                           DROP      2015-07-09:10:37:21
MGR_HIS                          DROP      2015-07-09:10:37:31
```

DROP TABLE table_name PURGE;

You cannot roll back a DROP TABLE statement with the PURGE clause, and you cannot recover the table if you drop it with the PURGE clause. This feature was not available in earlier releases.

Example:

```
Drop table emp purge;
```

You can purge the recycle bin with the following statement.

Example:

```
PURGE RECYCLEBIN;
```

15. External Table

An external table is a read only table whose metadata is stored in the database but whose data is stored outside the database.

Step-1

create a directory on unix name is 'ext'

$ mkdir ext

Step-2

create a text file in ext directory

]$cd ext

]$vi e1.ext

-enter the data in e1.txt

]$ cat e1.txt

100 riaz 10000

200 rash 20000

Step-3

]$export ORACLE_SID=riaz

]$sqlplus " / as sysdba"

Step-4

SQL> create or replace directory ext as '/home/oracle/ext';

Directory created.

Step-5

SQL> grant read,write on directory ext to public;

Grant succeeded

Step-6

```
create a table in database

{

  create table e_ext (     id    Number,      name    Varchar2(20),       salary
Number    )
```

organization external (type oracle_loader default directory ext access parameters

(records delimited by newline fields terminated by ',' missing field values are null) location ('e1.txt')) reject limit unlimited;

Step-7

```
SQL> select  *  from  e_ext;

        ID NAME                    SALARY
 ---------- ---------------------------------------- ----------

         100 riaz                        10000

        200 rash                        20000
```

how to view the external table information

desc USER_EXTERNAL_TABLES;

.log file

- all the process and syntax of creating a external table will be stored in log files E_EXTN_23347.log

.**bad file**

-if the data is not getting matched then it will move to the bad files. E_EXTN_23347.bad

[oracle@server1 rashmeet]$ cat E_EXTN_23347.bad

yujgkfgj

hjhjh

dhdh

Constraint Checking:

Constraints can have the following attributes :

- Deferrable or NOT Deferrable

- Initially Deferred or InitiallyImmediate

By default a constraint has a Not Deferrable attribute. We can not change this attribute at session level and transaction level, but if we created a constartint with Defferable attribute we can change it at session level as will as transaction level.

SQL> create table c1(id number primary key);

Table created.

```
SQL> SELECT CONSTRAINT_NAME,DEFERRABLE,DEFERRED FROM USER_CONSTRAINTS WHERE
TABLE_NAME='C1';

CONSTRAINT_NAME                DEFERRABLE            DEFERRED
------------------------------ --------------------- ---------------------------------------------

SYS_C003081                    NOT DEFERRABLE        IMMEDIATE

SQL> insert into c1 values(&I);

Enter value for i: 10

old 1: insert into c1 values(&I)

new 1: insert into c1 values(10)

1 row created.

SQL> /

Enter value for i: 20

old 1: insert into c1 values(&I)

new 1: insert into c1 values(20)

1 row created.

SQL> /

Enter value for i: 10

old 1: insert into c1 values(&I)

new 1: insert into c1 values(10)

insert into c1 values(10)

* ERROR at line 1: ORA-00001: unique constraint (RASHMEET.SYS_C003081) violated

SQL> select * from c1;

        ID

    ----------

        10

        20

SQL> create table c2(id number primary key deferrable);

Table created.

SQL> SELECT CONSTRAINT_NAME,DEFERRABLE,DEFERRED FROM USER_CONSTRAINTS WHERE
TABLE_NAME in ('C1','C2');
```

```
CONSTRAINT_NAME                DEFERRABLE     DEFERRED
------------------------------ -------------- ---------

SYS_C003081                    NOT DEFERRABLE IMMEDIATE

SYS_C003082                    DEFERRABLE     IMMEDIATE

SQL> insert into c2 values (&i);

Enter value for i: 10

old 1: insert into c2 values (&i)

new 1: insert into c2 values (10)

1 row created.

SQL> /

Enter value for i: 10

old 1: insert into c2 values (&i)

new 1: insert into c2 values (10)

insert into c2 values (10)

*

ERROR at line 1: ORA-00001: unique constraint (RASHMEET.SYS_C003082) violated.

SQL> select * from c2;

        ID

----------

        10
```

Note-- Defining Constraints at transaction level

```
SQL> set constraints SYS_C003082 deferred;

Constraint set.

SQL> insert into c2 values (&id);

Enter value for id: 10

old   1: insert into c2 values (&id)

new   1: insert into c2 values (10)

1 row created.

SQL> /

Enter value for id: 20
```

```
old   1: insert into c2 values (&id)

new   1: insert into c2 values (20)

1 row created.

SQL> /

Enter value for id: 20

old   1: insert into c2 values (&id)

new   1: insert into c2 values (20)

1 row created.

SQL> commit;

commit

* ERROR at line 1: ORA-02091: transaction rolled back

ORA-00001: unique constraint (RASHMEET.SYS_C003082) violated

SQL> select * from c2;

no rows selected.
```

Note-- now transaction level setting will remove because of commit command. now it will work as initially immediate.

```
SQL> insert into c2 values (&id);

Enter value for id: 10

old   1: insert into c2 values (&id)

new   1: insert into c2 values (10)

1 row created.

SQL> /

Enter value for id: 10

old   1: insert into c2 values (&id)

new   1: insert into c2 values (10)

insert into c2 values (10)

*

ERROR at line 1: ORA-00001: unique constraint (RASHMEET.SYS_C003082) violated
```

Note-- Defining Constraints at session level.

```
SQL> alter session set constraints = deferred; Session altered.
```

Note : Here all Deferrable constraints will become deferred.

```
SQL> select * from c2;

        ID

----------

        10

SQL> insert into c2 values (&id);

Enter value for id: 20

old   1: insert into c2 values (&id)

new   1: insert into c2 values (20)

1 row created.

SQL> /

Enter value for id: 10

old   1: insert into c2 values (&id)

new   1: insert into c2 values (10)

1 row created.

SQL> commit;

commit

* ERROR at line 1:

ORA-02091: transaction rolled back

ORA-00001: unique constraint (RASHMEET.SYS_C003082) violated

SQL> select * from c2;

no rows selected

SQL> alter session set constraints=default;

Session altered.

SQL>  insert into c2 values (&id);

Enter value for id: 10

old   1:  insert into c2 values (&id)

new   1:  insert into c2 values (10)

1 row created.

SQL> /

Enter value for id: 10
```

```
old   1:  insert into c2 values (&id)

new   1:  insert into c2 values (10)

insert into c2 values (10)

*

ERROR at line 1:

ORA-00001: unique constraint (RASHMEET.SYS_C003082) violated
```

Note--data dictionary will not update in transaction level and session level setting

```
SQL>   SELECT  CONSTRAINT_NAME,DEFERRABLE,DEFERRE   FROM  USER_CONSTRAINTS  WHERE
TABLE_NAME in ('C1','C2');

CONSTRAINT_NAME                DEFERRABLE     DEFERRED
------------------------------ -------------- ---------

SYS_C003081                    NOT DEFERRABLE IMMEDIATE

SYS_C003082                    DEFERRABLE     IMMEDIATE
```

Note-- Defining Constraints permanentaly

```
SQL> alter table c2 drop primary key;

Table altered.
```

note-- for this setting need to drop the old constraint.

```
SQL> alter table c2 add primary key(id) DEFERRABLE INITIALLY DEFERRED;

Table altered.

SQL>  SELECT  CONSTRAINT_NAME,DEFERRABLE,DEFERRED   FROM  USER_CONSTRAINTS  WHERE
TABLE_NAME='C2';

CONSTRAINT_NAME                DEFERRABLE     DEFERRED
------------------------------ -------------- ---------

SYS_C003083                    DEFERRABLE     DEFERRED

SQL> insert into c2 values(&i);

Enter value for i: 10

old   1: insert into c2 values(&i)

new   1: insert into c2 values(10)

1 row created.

SQL> /

Enter value for i: 20
```

```
old   1: insert into c2 values(&i)
new   1: insert into c2 values(20)
1 row created.
SQL> /
Enter value for i: 10
old   1: insert into c2 values(&i)
new   1: insert into c2 values(10)
1 row created.
SQL> commit;
commit
*
ERROR at line 1:
ORA-02091: transaction rolled back
ORA-00001: unique constraint (RASHMEET.SYS_C003083) violated
SQL> select * from c2;
        ID
----------
        10
```

Note-- we can change the permanent setting of either immediate or deferred in deferrable

```
SQL> set constraints SYS_C003083 immediate;
Constraint set.
SQL>  insert into c2 values(&i);
Enter value for i: 10
old  1:  insert  into  c2  values(&i)
new  1:  insert  into  c2  values(10)
insert into c2 values(10)
*
ERROR at line 1:
ORA-00001: unique constraint (RASHMEET.SYS_C003083) violated
```

16. Assignments

1. if exp1 and exp2 are not equal it should print first exp if both exp are same then it will display 'no_match' string.output should be in this way.

   SQL> ex1 ex2 Result

   ---------- ---------- ----------

   5 11 5

   6 5 6

   7 5 7

   7 6 7

   5 5 no_match

   5 7 5

   6 6 no_match

   7 7 no_match

   5 5 no_match

2. Find those employees whose last name ends with n letter. (dont use like operator)

3. Display the output. date, hour, min, day--

   date hour min day

   --- --- ---

   22-feb-11 05:49:42 05 49 TUESDAY

4. Round these dates in month and year.

   -15-feb-11

   -march-12-11

   -2011-june

   -21 -04-2011-oct

5. use the hire_date column from employees table and prove that yy works on current century and rr uses internal algorithm. display the output.

   rr-date yy-date

   ---------- ----------

   11-jul-1998 11-jul-2098

19-dec-1999 19-dec-2099

04-feb-1996 04-feb-2096

03-mar-1997 03-mar-2097

6. display the output. (use sysdate function from dual table)

SQL> seventeen of 10th

7. convert this value '$20,000.00' in number. (20000)

8. add 5 days in 'march-2011-12'

9. find the no of days between '12-feb-2011' and 'march-12-1999'.

10. findout the day for the date ('15-jan-11')-- use dual table

11. prompt the value for date and it should display the day for prompted date.

12. Display Region and total number of employees from that region in the following format

East	West	North	South
10	40	34	76

13. replace AA where a in the last_name. display the output..

NAME	LAST_NAME
TAAylor	Taylor
TobiAAs	Tobias
TuvAAult	Tuvault
UrmAAn	Urman

14. dispaly the output.first letter from the first_name column and then aad '_' and then last_name column then '@focus.com' string.

J_Urman@focus.com

P_Vargas@focus.com

C_Vishney@focus.com

S_Vollman@focus.com

A_Walsh@focus.com

15. display the employee number, hire date, number of years employed, six-month review date, first Friday after hire date, and last day of the hire month for all employees who have been employed for more then 15 years.

ID	HIRE_DATE	TENURE	REVIEW	first_FRI	Last_DAY
109	16-AUG-94	16	16-FEB-95	19-AUG-94	wednesday
114	07-DEC-94	16	07-JUN-95	09-DEC-94	Wednesday

16. if employee does not report to a manager then print 'no manager' else print 'reports to manager and manager_name'.

LAST_NAME	MGR_NAME
Urman	reports to ==Greenberg
Vargas	reports to ==Mourgos
Vishney	reports to ==Errazuriz
Vollman	reports to ==King

17. Display those employees who are getting more then '$15000,00'. i have to compare the salary column in this format ('$15000.00') only.
18. Display those employees whose hire_date is greater then march-1999-01. i have to compare hire_date column in this format ('march-1999-01') only, but display the output in this way.

HIRE_DATE

twenty-three-november of nineteen ninety-nine

nineteen-march of nineteen ninety-nine

twenty-four-january of two thousand

twenty-three-february of two thousand

twenty-four-march of two thousand

19. Dsplay total no. of employees.
20. Display employees's first name and last name and sort them by last_name.
21. Display full name of employees and full name his manager.
22. Display a list full name of employees and their salaries . Display only top 5 employees.
23. Display employees's full name and the department of that employees.

24. Display list of departments and total number of employees in it.
25. Display list of departments and employees in that list even if a department does not have any employee in it.
26. Display the list of departments and total salary of that department.Display only list of departments whose total salary is more than 50000.
27. Display list of employees (employee full name and department name) who are not from the accounting department.
28. Display employees name,salary, job_title of employees who were hired in the year of 1994.
29. Which country has the maximum no of employees.
30. Display the employee_id, last_name, and salary increased by 15.5%(expressed as a whole number) for each employee. Lable the column New Salary. And then add a column that substract the old salary from the new salary. Lable the column increase.
31. Increase the salary range of all jobs from executive department by 20%..
32. Display a list of departments and their managers along with their phone_number. Do not display country code of phone_number.
33. List of the employees and their salaries whose salary is less than the average salary of the company.
34. List of the employees and their salaries whose salary is more than the average salary of the company.
35. Display the list of employees who have a and e in their name.
36. Display the department_name,location,lastname, job_id salary of those employees who work in a specific location. Prompt the user for the location.
37. Display the jobs that are found in the Administration and Executive departments.Also display the no of employees for these jobs. Show the job with the highest no of employee first.
38. Show all employees who were hired in the first half of the month(before the 16th of the month).
39. Display the No Commission where commission_pct is null.
40. Create a table named dept using departments table, and truncate dept and then use the merge statement to get the data back.
41. Create two tables emp1 and dept1 and establish a relationship between both tables.
42. Drop the table employees and the get back it using FLASHBACK statement from recyclebin.
43. Produce a company organization chart that shows a management hierarchy. Start with the person at the top level, exclude all those job ID IT_PROG, and exclude De Haan and those employees who report to De Haan.
44. Create a table EMP_NEW and col name should be (EMP_NO,NAME,and SALARY), apply the constraint on salary column that salary should not be less then 1000.
45. Display City and total no. of employees in that city. List only top 5 cities

46. Delete any 10 rows from employee.
47. Display total number of employees from the Acounting Department
48. Display Department name and total number of employees
49. Display region and total number of employees from that region
50. Display City and total no. of employees in that city. List only top 5 cities.

17. Interview Question

1) What is the difference between truncate and drop?
2) What is the difference between truncate and delete?
3) How do you delete rows but keep one row?
4) What is deadlock.How to resolve deadlock?
5) Which version of oracle are you using how to find out?
6) Display list of tables from ahmed's or any user's schema?
7) Show all the employees who were hired on he day of the week on which the highest number of employees were hired.
8) Display list of schema and total number of tables in each schema. Sort the list from the largest number of tables to lowest number of tables.
9) How would you increase the performance of the following select statement –

   select * from sales_history where customer_id =288309
10) How will you identify duplicate rows in a table?
11) Get the details from employees table whose joining month is January.?
12) What is the difference between UNION and UNION ALL?
13) what is an execution plan?
14) How do you find out an execution plan of a select statement?
15) Can you have 2 primary keys on one table?
16) What is a composite primary key?
17) How to find out what is the primary key of a table?
18) How to find out if a table has any indexes and on what columns?
19) Is select statement a DDL or DML?
20) What is the difference between ddl and dml?
21) What is locking?
22) Does select statement lock the rows?
23) What kind of lock is acquired by update or delete?
24) Explain what is a savepoint ? In what scenario would you use it? Give an example.

25) What is the use of cascade constraint?

26) What is the meaning of a transaction? Give an example.

27) Find the third max salary from the employees table?

28) Display all employees whose employee_id is an odd number?

29) List dept no., Dept name that do not have any employees in it. Write this query using 2 different ways

30) Display department name and total number of employees in that department. Display only top 5 by total number.

31) Display Department name and total number of employees in that department. List departments even if they do not have any employees in it.

32) Select all records where last_name starts with 's' and its length is 6 char.

33) Display employee name, his/her department name, his/her manager name. Sort it by Department Name from a-z

34) If there are two tables tab1 and tab2 with identical structure. How will you find out employees that are in tab1 and also in tab2.

35) If there are two tables tab1 and tab2 with identical structure.

How will you find out employees of tab1 except those employees of tab1 whoever present in tab2.

36) Display list of departments and total salary of that department. Display only list of departments whose total salary is more than 50000.

37) Display employee names, salary, jobs of employees who were hired in the year of 2000

38) Display all regions from the regions tables and total number of employees in that region. List only those regions that have more than 12 number of employees. Arrange the list from highest employee number to lowest employee number

39) Display list of tables in your schema ?

40) Display list of objects in ahmed's schema?

41) How do you find out the current date?

42) How do you find out the current time?

43) How to find out the foreign keys on any table?

44) How do you change the sql prompt in sqlplus?

45) How do you create a materialized view?

46) How do you refrest a MV?

47) What is the difference between fast and complete refresh?

48) What is a materialized view log?

49) What is a difference between materialized view and a regular view?

50) T/F - Data in Oracle is case sensitive?

51) T/F - SQL commands are case sensitive.?

52) How do find out the time taken for executing a select statement in sqlplus.

53) What is an escape character? Give an example.

54) List total number of employees who are from Americas region and whose salary is more than 4000

55) Display total number of employees from the Europe region who had worked in the Sales department in the year 1998.

56) List of employees and their salaries whose salary is less that the average salary of the company.

57) List employees who earn more that the average salary of their department?

58) Display list of schema and total number of tables in each schema. Sort the list from the largest number of tables to lowest number of tables.

59) Get names of employees from ESC table who has '%' in Last_Name?

60) Which country has the maximum number of employees?

61) List of employees and their salary who are getting paid below 50% of their salary range.

62) Display a list of departments and managers' name along with their phone numbers. Do not display country code of phone number.

63) Write a query that will identify all employees who work in departments located in the United Kingdom and whose salary is more than the average salary of their department.

64) What are the most common steps you have encountered in an execution plan?

65) Increase the salary range of all jobs from the Executive department by 20%.

66) Have you used a Database Link. How, Where. Give Example?

67) What is a difference between Primary key and Unique Index?

68) How do you find out if there are any rows that are locked in a table?

69) What is the difference between views and synonym?

70) Why do we need public synonym?

71) What is private synonym and how to create it?

72) What are the scenarios if user having simple view and still not able to insert any row in base table?

73) Why do we need views?are views physically present in database?

74) How do some one know that how many constraints are on any table?

75) Create a query that displays salary,employee_id and indicates the amounts of their annual salaries with asterisks. Each asterisk signifies a thousand dollars. If the salary>1000 then salary should be padded with asterisks else the original salary?

76) I have one table which is having certain data in it what will happen if I want to apply primary key at this level?

77) What are the new fetures of 12C?

78) What is listagg function,rank and dense rank in sql?

79) What is inline view?

80) What is global sequence and session sequence in 12C?

81) Difference between 1g and 12C?

82) What are the operator precedence?

83) Suppose you are having a table t1 whose structure and values are as

Id	name
1	A
2	B
3	C
4	D

You have to write a select statement so that the output of the query will be

Id	name
1	A
2	BB
3	CCC
4	DDDD

84) How many data dictionary you know.Name some important one?and there types also?

85) What is flashback?explain in reference of rollback?

86) What are the differences among ROWNUM, RANK and DENSE_RANK?

87) What is the difference between ROWNUM pseudo column and ROW_NUMBER() function?

88) What is self join and why it is required?

89) What is the difference between UNION,MINUS and INTERSECT?

90) What is the difference between WHERE and HAVING clause?

91) What is the difference between JOIN and UNION?

92) What is the difference between UNION and UNION ALL?

93) What is the difference between INNER and OUTER JOIN?

94) What is constraints? Tell abt its level?

95) What is ACID property in database?

96) What are the different types of statements in sql?

97) What are the differences between primary,foreign and unique constraints?

98) Can a table have multiple primary key,foreign key and unique key?

99) What is an index? How a database index can help performance?

Questions Based On Sales Model.

Q1: Who are the top 5 customers?

Q2: Who are the top 5 sales people?

Q3: Which is the 2nd best selling product?

Q4: Display product name, category of that product that are shipped by Speedy Express.

Q5: Display list of cities where Dairy Products is not being sold?

Q6: Which is the best performing year (sales wise)?

Q7: What is the average delay (in days) for shipping orders in the month of January?

Q8: Display region and the total sales for that region for Q1 and Q2?

Q9: List of customers whose average yearly sale is more that Rs. 10000?

Q10: Top 5 products which have largest inventory - Rs wise?

Q11: Top 5 products which have largest inventory - Quantity wise?

18. Additional Features in 12C

Identity Columns

A column in a table can be marked as identity column which generates its value by itself. Oracle implicitly creates a sequence of default configuration for the identity column. For each insert operation, the current value of the sequence gets automatically assigned to the identity column.

The feature syntax is as below –

```
SQL>create table t_id_col
  2  ( x number
  3      generated by default
  4          as identity
  5          ( start with 10
  6            increment by 15 )
  7          primary key,
  8    y varchar2(30)
  9  )
 10  /
```

Now we insert some rows into table t_id and see the result.

```
SQL> insert into t (x,y) values ( 1, 'hello1' );
1 row created.

SQL> insert into t (x,y) values ( default, 'hello2' );
1 row created.

SQL> insert into t (y) values ( 'hello3' );
1 row created.

SQL> select * from t;

         X  Y
__________  __________
         1  hello1
        10  hello2
        25  hello3
```

Extended String Datatypes

Until Oracle 11g SQL, the maximum precision allowed for a string type column was 4000. In Oracle 12c, the precision has been increased upto 32767 bytes or 32K. The new string data types will be known as Extended String Types in Oracle 12c. The feature is controlled by an initialization parameter MAX_STRING_SIZE. The database must be in upgrade mode to enable this feature. Note that once the feature is enabled, thereafter the parameter cannot be disabled.

ALTER SYSTEM SET max_string_size = ENABLED;

Invisible Columns

This new feature in 12C allows to make invisible columns for generic queries, operations. To make invisible column you need to use INVISIBLE clause

Following operations don't see invisible columns

```
SELECT * FROM statements in SQL
DESCRIBE commands in SQL*Plus
%ROWTYPE attribute declarations in PL/SQL
Describes in Oracle Call Interface (OCI)
```

Invisible can be declared during table creation

```
create table test_tbl
 (  id  number,
    id1 number INVISIBLE,
    id2 number,
    id3 number,
    id4 number
 );
```

or by alter table

alter table test_tbl modify(id2 INVISIBLE);

generic operations donâ€™t see INVISIBLE columns

```
desc test_tbl;
```

```
      Name Null Type
      ---- ---- ------
      ID         NUMBER
      ID3        NUMBER
      ID4        NUMBER
```

so backup using SELECT * FROM doesn't copy INVISIBLE columns

```
create test_tbl_bkp
as
select * from test_tbl;

desc test_tbl_bkp;
```

```
      Name Null Type
      ---- ---- ------
      ID         NUMBER
      ID3        NUMBER
      ID4        NUMBER
```

but still you can access INVISIBLE column explicitly

```
create table test_tbl_bkp1
as
select id, id1, id2, id3, id4 from test_tbl;
desc test_tbl_bkp1;
```

```
Name Null Type
---- ---- ------
ID   NUMBER
ID1  NUMBER
ID2  NUMBER
ID3  NUMBER
ID4  NUMBER
```

```
insert into test_tbl  (id, id1, id2, id3, id4) values(1,1,1,1,1);

commit;

select id, id1, id2, id3, id4 from test_tbl;
```

ID	ID1	ID2	ID3	ID4
1	1	1	1	1

INVISIBLE has got impact on ordering column_id USER|ALL|DBA_TAB_COLUMNS. For all INVISIBLE columns column_id is null.

```
select table_name, column_name, column_id
from user_tab_columns where table_name='TEST_TBL';
```

TABLE_NAME	COLUMN_NAME	COLUMN_ID
TEST_TBL	ID4	3
TEST_TBL	ID3	2
TEST_TBL	ID2	
TEST_TBL	ID1	
TEST_TBL	ID	1

each modification from **INVISIBLE to VISIBLE** will put modified column as last for ordering.

alter table test_tbl modify(id2 VISIBLE);

```
select table_name, column_name, column_id
from user_tab_columns where table_name='TEST_TBL';
```

TABLE_NAME	COLUMN_NAME	COLUMN_ID
TEST_TBL	ID4	3
TEST_TBL	ID3	2
TEST_TBL	ID2	4
TEST_TBL	ID1	
TEST_TBL	ID	1

DEFAULT ON NULL

DEFAULT ON NULL in Oracle 12c assign default value if INSERT attempts to assign a value that evaluates to NULL.

Syntax:

DEFAULT ON NULL <VALUE>

Example:

```
drop table test_tbl;

create table test_tbl
( id  number default on null 5,
  id1 number);

insert into test_tbl values(null, 10);
insert into test_tbl values(100, 20);


select * from test_tbl;

        ID         ID1
---------- ----------
         5          10
       100          20
```

To remember

NOT NULL constraint and NOT DEFERRABLE constraint state are implicitly specified when DEFAULT NOT NULL is used

select dbms_metadata.get_ddl('TABLE', 'TEST_TBL') from dual;

```
CREATE TABLE "TOMASZ"."TEST_TBL"
( "ID" NUMBER DEFAULT 5 NOT NULL ENABLE,
  "ID1" NUMBER
)


select constraint_name, constraint_type, deferrable, search_condition
from user_constraints
where table_name='TEST_TBL';

CONSTRAINT_NAME    CONSTRAINT_TYPE    DEFERRABLE               SEARCH_CONDITION
------------------ ------------------ ------------------------ ----------------
SYS_C0010162        C                 NOT DEFERRABLE          "ID" IS NOT NULL
```

APPROX_COUNT_DISTINCT function

The **APPROX_COUNT_DISTINCT** function is available starting with Oracle Database 12c Release 1 (12.1.0.2).

It returns approximate number of rows that contain distinct values of expr.

Syntax:

APPROX_COUNT_DISTINCT(expr)

For expr, you can specify a column of any scalar data type other than BFILE, BLOB, CLOB, LONG, LONG RAW, or NCLOB.

This function provides an alternative to the COUNT (DISTINCT expr) function, which returns the exact number of rows that contain distinct values of expr. APPROX_COUNT_DISTINCT processes large amounts of data significantly faster than COUNT, with negligible deviation from the exact result.

APPROX_COUNT_DISTINCT ignores rows that contain a null value for expr. This function returns a NUMBER.Examples

The following statement returns the approximate number of rows with distinct values for manager_id:

```
SELECT APPROX_COUNT_DISTINCT(manager_id) AS "Active Managers"
  FROM employees;


Active Managers
---------------
             18
```

The following statement returns the approximate number of distinct employees for each departments.

```
SELECT department_id, APPROX_COUNT_DISTINCT(employee_id) AS "total"
 FROM employees
 GROUP BY department_id
 ORDER BY department_id;
```

Truncate table CASCADE

In the previous releases, there wasn't a direct option provided to truncate a master table while it is referred to by the child tables and child records exist. The TRUNCATE TABLE with CASCADE option in 12c truncates the records in the master table and automatically initiates recursive truncate on child tables too, subject to foreign key reference as DELETE ON CASCADE. There is no CAP on the number of recursive levels as it will apply on all child, grand child and great grandchild etc. This enhancement gets rid of the prerequisite to truncate all child records before truncating a master table. The new CASCADE clause can also be applied on table partitions and sub-partitions etc.

Syntax:

```
TRUNCATE TABLE TABLE_NAME CASCADE;
```

An **ORA-14705** error will be thrown if no ON DELETE CASCADE option is defined with the foreign keys of the child tables.

ROW limiting for Top-Nresult queries

There are various indirect approaches/methods exist to fetch Top-N query results for top/bottom rows in the previous releases. In 12c, retrieving Top-N query results for top/bottom rows simplified and become straight forward with the new FETCH FIRST|NEXT|PERCENT clauses.

Syntax:

FETCH FIRST|NEXT|PERCENT

In order to retrieve top 10 salaries from EMPLOYEES table, use the following new SQL statement.

```
SQL> SELECT eno,ename,sal FROM employees ORDER BY SAL DESC

          FETCH FIRST 10 ROWS ONLY;
```

The following example fetches all similar records of Nth row. For example, if the 10th row has salary of 5000 value, and there are other employees whose salary matches with the Nth value, the will also be fetched upon mentioning WITH TIES clause.

```
SQL> SELECT eno,ename,sal FROM employees ORDER BY SAL DESC

     FETCH FIRST 10 ROWS ONLY WITH TIES;
```

The following example limits the fetch to 10 per cent from the top salaries in the EMPLOYEES table.

```
SQL> SELECT eno,ename,sal FROM employees ORDER BY SAL DESC

     FETCH FIRST 10 PERCENT ROWS ONLY;
```

The following example offsets the first 5 rows and will display the next 5 rows from the table:

```
SQL> SELECT eno,ename,sal FROM emp ORDER BY SAL DESC

     OFFSET 5 ROWS FETCH NEXT 5 ROWS ONLY;
```

All these limits can be very well used within the PL/SQL block too.

```
BEGIN

           SELECT sal BULK COLLECT INTO sal_v FROM EMP

                      FETCH FIRST 100 ROWS ONLY;

END;
```

READ Priviledge

New privileges READ, READ ANY TABLE have appeared in Oracle Database 12c. They are available since release 12.1.0.2. They work almost the same as standard SELECT and SELECT ANY TABLE except SELECT and SELECT ANY TABLE can do additionally

--acquires exclusive lock on a table LOCK TABLE IN EXCLUSIVE MODE;

--acquires row lock on a table rows SELECT * FROM table FOR UPDATE;

So simple grant SELECT lets users to make harm(can block other users) in a database just by executing above commands. Itâ€™s time to start using READ and READ ANY TABLE to avoid it and improve security. You can grant/revoke them to the same objects like

SELECT:*tables*views*materializedviews*synonyms.

EXAMPLE:

```
GRANT READ ON emp TO ahmed;

GRANT READ ON emp_view TO ahmed;

GRANT READ ANY TABLE TO ahmed;

REVOKE READ ON emp_view FROM ahmed;

REVOKE READ ANY TABLE FROM ahmed;
```