Akhil (Admin)
Mobile: 9154156192 (Only Whatsapp)

Notes link:
bit.ly/oracledbnotes

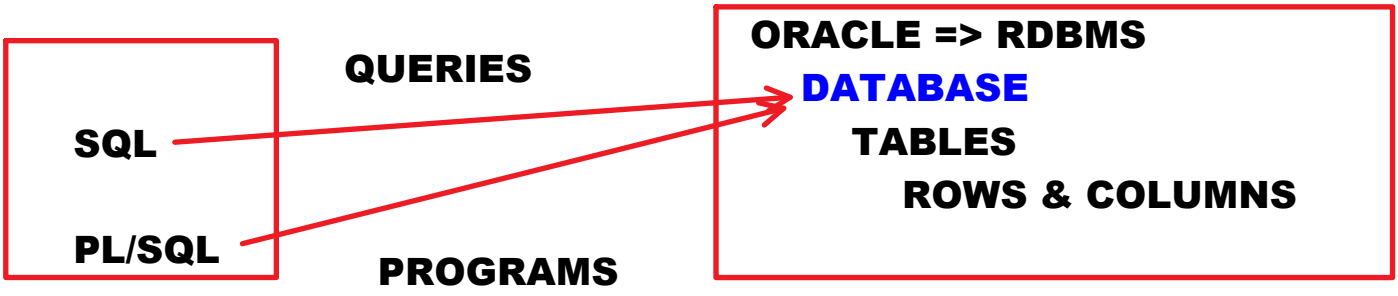
ORACLE 21C Software Link:
bit.ly/oracle21csoftware

Oracle Installation Video Link:
bit.ly/oracle21cinstallation

Oracle [SQL & PL/SQL] @ 7:30 AM (IST) by Mr.Shiva Chaitanya
Day-1 https://youtu.be/vh8z5yO4zEA
Day-2 https://youtu.be/eqn7FJ2BEqk
Day-3 https://youtu.be/GDB72pOuGGM
Day-4 https://youtu.be/C0IjGE74hoE
Day-5 https://youtu.be/t2cud9FKhdk
Day-6 https://youtu.be/C4zTm4RD0o8
Day-7 https://youtu.be/SmxcUsPNfww

Syllabus

Tuesday, June 11, 2024 7:43 AM



SQL:

TABLES

SQL Commands	DDL, DRL, DML, TCL, DCL
Built-In Functions	max() min() lower() upper() ...
Clauses	Group By, Having, Order By ...
Joins	Inner join, Outer join .....
Sub Queries	Non-correlated, correlated
Set operators	union, union all, intersect ...
Constraints	Primary key, foreign key, check ....
VIEWS	
SEQUENCES	
INDEXES	
MATERIALIZED VIEWS	
SYNONYMS	

PL/SQL:

PL/SQL Basics	**data types, declare, assign, print, read Using SQL commands in PL/SQL**
Control Structures	**IF .. THEN, FOR, WHILE ....**
CURSORS	
COLLECTIONS	
EXCEPTION HANDLING	
Stored Procedures	
Stored Functions	
Packages	
Triggers	
Working with LOBs	
Dynamic SQL	

ORACLE

Thursday, June 6, 2024 7:44 AM

data store => is a location
database => data store => is a location. in that organization's business data stored permanently

DBMS => is a software => used to maintain the database
RDBMS => is a software => used to maintain the DB in the form of tables

Metadata

run the business
analyze the business

BANK

Branches
Customers
Transactions
Products
Employee

customer is depositing amount
customer is withdrawing amount
opening account
closing account

2020 ?
2021 ?
.
.
2024 ?

Amazon

Products
Customers
Orders
Payments
Wishlist
Sellers
Suppliers

searching for products
placing orders
adding items to wishlist

2020 ?
2021 ?
.
.

.

2024 ?

Goal:
Storing organization's business data permanently in computer

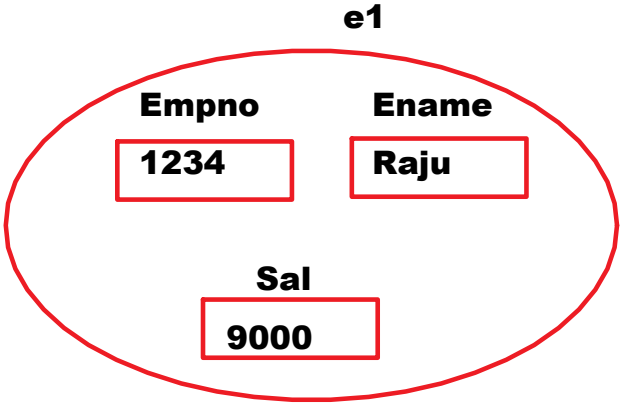
Variable:

- **Variable is temporary.**

int empno;

Object:

- **Object is temporary.**



File:
File is permanent

Database:
Database is permanent

File	Database
• suitable for 1 user • less security • suitable to store small amounts of data	• suitable for multiple users • more secured • suitable to store large amounts of data

Data Store:

- is a location where data is stored.

Example:
File, Database

Database:

- is a kind of data store.
- Database is a location where organization's business data stored permanently.

Example:

BANK DB	COLLEGE DB
Branches Customers Transactions Products Employee	COURSES STUDENTS FEE MARKS STAFF

- Database is a collection of interrelated data in an organized form.

interrelated =>

BANK DB contains bank related data. not college related data

organized => arranging in specific way [table]

DBMS:

- **DBMS => Database Management System / Software**
- **DBMS is a software that is used to create and maintain the database.**

Evoluation of DBMSs:

Before 1960s => BOOKS

In 1960s => FMS [File Management Software]

In 1970s => Hierarchical DBMS [HDBMS]
Network DBMS [NDBMS]

In 1976 => Relational DBMS Concept
E.F.Codd

ORACLE COMPANY FOUNDER => Larry Ellison

In 1977 => Software Development Laboratories

In 1979 => renamed company name as Relational Software Inc
introduced ORACLE software

In 1983 => renamed company name as ORACLE corp.

RDBMS:

- **RDBMS => Relational Database Management System/ Software.**
- **It is a kind of DBMS.**
- **Relation => Table**

- **RDBMS is a software that is used to create and maintain the database in the form of tables.**

Examples:

ORACLE, SQL SERVER, MY SQL, DB2, Postgre SQL

Table:

- **Table is a collection of rows and columns**
- **row is horizontal representation of data**
- **column is vertical representation of data**

Example:

EMPLOYEE → **Table / Relation / Entity**

EMPID	ENAME	SAL
1234	Kiran	12000
1235	Vijay	15000

→ **Column / Attribute / Property / Field**

↓ **row / tuple / entity instance / record**

Metadata / Data Definition:

- **Metadata is the about the data.**
- **It describes about the data.**

Examples:

Column name, Table name, Data Type, Field size

Example:

EMPLOYEE

EMPID NUMBER(4)	ENAME VARCHAR2(10)	SALARY
1234	Ravi	9000
1235	Kiran	7000
Ramu ERROR		
25-DEC-23 ERROR		
9999		
10000 ERROR		

ORACLE

Friday, June 7, 2024 8:06 AM

ORACLE:

- **is a Relational Database Management Software [RDBMS]**

- **it is used to create and maintain the database in the form of tables.**
  database = organization's business data

- **Using ORACLE software we can store, manipulate and retrieve the data of database.**

  manipulate => insert / update [modify] / delete

  customer opened account => INSERT
  customer is withdrawing amount => UPDATE
  customer is closing account => DELETE

  emp joined => INSERT
  emp promoted => UPDATE
  emp resigned => DELETE

Retrieve => opening existing data

searching for products
checking balance
transactions statement

- **ORACLE software 2nd version introduced in 1979.**
  1st version they didn't release to market.

- **For LINUX OS latest version is: ORACLE 23AI**
- **For WINDOWS OS latest version is: ORACLE 21C**

Evaluation of DBMSs:

Before 1960s => BOOKS

In 1960s => FMS [File management Software]

In 1970s => Hierarchical DBMS [HDBMS]
=> Network DBMS [NDBMS]

In 1976 => RDBMS concept introduced => E.F.Codd

ORACLE company founder => Larry Ellison

In 1977 => Software Development Laboratories

In 1979 => Relational Software Inc.
introduced ORACLE software 2nd version

In 1983 => renamed company name as: ORACLE corp.

Examples of RDBMS:

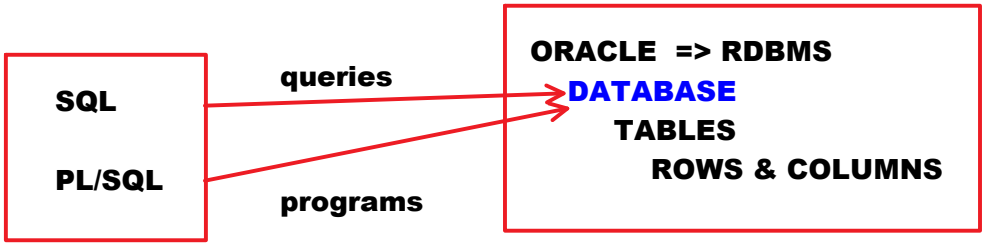
ORACLE => ORACLE company

SQL SERVER => Microsoft

DB2 => IBM

MY SQL => Sun Micro Systems
ORACLE

Postgre SQL => Postgre Forum [a group of companies]



To communicate with ORACLE DATABASE we can use 2 languages. They are:

- **SQL => query language => queries**
- **PL/SQL => programming language => programs**

SQL:

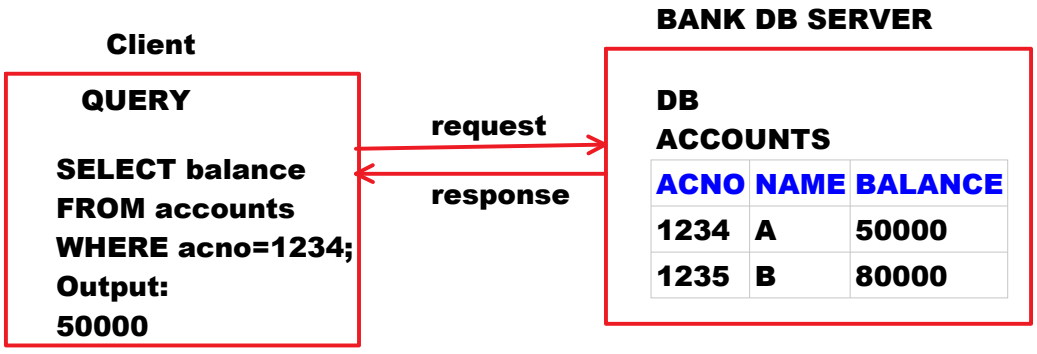
- SQL => Structured Query Language.

- It is a query language.
- In this we write queries to communicate with ORACLE DB.

- Query => request / command / instruction

- Query is a request that is sent to DB Server.

Example:
SELECT ename, sal FROM emp; => QUERY



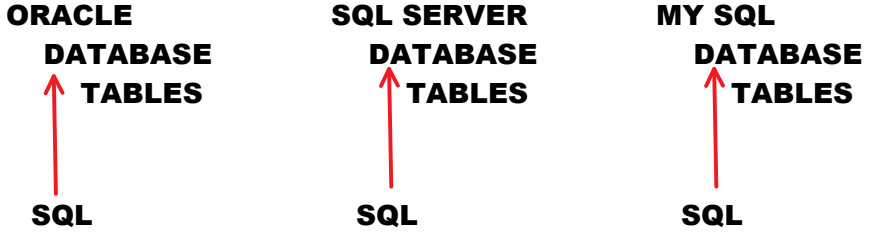
- SQL is Non-Procedural Language.
  In SQL, we will not write any set of statements or programs. Just we write QUERIES.

- SQL is Unified Language. It is common language to work with many relational database management softwares.

In C:
Function:
a set of statements

In Java:
Method:
a set of statements

In PL/SQL:
Procedure:
a set of statements
Sub program



- SQL provides commands to write the queries
  Examples:

SELECT, INSERT, UPDATE, DELETE

- **SQL provides Functions to make our actions easier.**
  Examples:
  max(), min()

- **SQL provides Operators to perform operations.**
  Example:
  *** , + , > , BETWEEN AND, IN**

- **SQL provides JOINS concept to retrieve the data from multiple tables.**

DEPT

DEPTID	DNAME
10	HR
20	SALES

EMP

EMPID	ENAME	SAL	DEPTID
1234	A	12000	20
1235	B	20000	10

JOINS

ENAME	DNAME
EMP	DEPT

PL/SQL:

- **PL => Procedural Language**
- **SQL => Structured Query Language**

- **It is programming language.**

- **In this, we develop the programs to communicate with ORACLE DATABASE.**

- **It is a PROCEDURAL LANGAUGE. In this we define a set of statements or programs.**

- **PL/SQL = SQL + Programming**

- **PL/SQL is extension of SQL. All SQL QUERIES can be written as statements in PL/SQL program.**

BANK DB

Branches table

IFSC_Code	City	State	Country

Customers

Custid	CNAME	AADHARNUM	PAN_NUM	ACNO

Transactions

Acno	date_time	ttype	amount

Employee

EMPNO	ENAME	SAL

SQL Commands

Monday, June 10, 2024 8:50 AM

ORACLE SQL provides 5 sub languages. Every sub language provides commands. They are:

SALARY => metadata

6000 => data

DDL: • Data Definition Language • Data Definition => Metadata • it deals with metadata	create alter drop flashback purge truncate rename
DRL / DQL: • DRL => Data Retrieval Language • DQL => Data Query Language • it deals with data retrievals	select
DML: • Data Manipulation Language. • It deals with data manipulations.	insert update delete insert all merge
TCL: • Transaction Control Language • It deals with transactions.	commit rollback savepoint
DCL / ACL: • DCL => Data Control Language • ACL => Accessing Control Language • It deals with data accessibility	grant revoke

DDL

Tuesday, June 11, 2024 8:03 AM

DDL:

- **DDL => Data Definition Language**
- **Data Definition => metadata.**
- **It deals with metadata**
- **metadata is the data about the data**

SALARY => metadata

6000 => data

ORACLE SQL provides following DDL commands:

- **create**
- **alter**
- **drop**
- **flashback**
- **purge**
- **truncate**
- **rename**

ORACLE DB OBJECTS:

Table
View
Index
Materialized View
Sequence
Synonym
Procedure
Function
Package
Trigger

Create:

- **Create command is used to create the ORACLE DB OBJECTS like tables, view, indexes ... etc.**

Syntax to create the table:

```
CREATE TABLE <table_name>
(
    <column_name> <data_type> [,
    <column_name> <data_type> ,
    .
    .]
);
```

[ ]	optional
< >	any

For Windows OS, latest version is ORACLE 21C.
For LINUX OS, latest version is ORACLE 23AI.

Till ORACLE 21c => max of 1000 columns
In ORACLE 23AI => max of 4096 columns

Data Types of SQL:

Character Related **'RAJU'** **'MANAGER'** **'B.tech'**	**Char(n)** **Varchar2(n)** **LONG** **CLOB** **nChar(n)** **nVarchar2(n)** **nCLOB**
Integer Related **1234** **78** **17**	**Number(p)** **Integer** **Int**
Floating Point Related **68.56** **1280.75**	**Number(p,s)** **Float** **Binary_Float** **Binary_Double**
Date & Time Related **25-DEC-23** **11-JUN-24 8:30:15.123456 AM**	**Date** **Timestamp**
Binary Related **images, audios, videos**	**BFILE** **BLOB**

string => group of characters
'raju' 'hyd'

Char(n):

- **n => max no of chars**
- **It is used to hold strings.**
- **it is fixed length data type.**
- **it is used to hold fixed length chars.**
- **max size: 2000 Bytes [2000 chars]**
- **default size: 1**

Varchar2(n):

- **n => max no of chars**
- **It is used to hold strings.**
- **It is variable length data type.**
- **it is used to hold variable length chars.**
- **max size: 4000 Bytes [4000 chars]**
- **default size: there is no default size**

State Code CHAR(2) **ENAME VARCHAR2(10)**

State_Code CHAR(2)	ENAME VARCHAR2(10)
TG	Raju
AP	Sai
MH	Naresh

Note:
character related data types can hold letters, digits and special symbols.

Vehicle_Number CHAR(10)	mail_id VARCHAR2(30)
TG09AA1234	ravi123@gmail.com
	vijay98765_kumar@yahoo.com

PAN_CRAD_NUM CHAR(10)	job VARCHAR2(20)
ABCDE1234F	Manager
	Full Stack Developer

Note:
VARCHAR2 data type can hold max of 4000 chars only.
To hold more than 4000 chars we use LONG or CLOB.
LONG has some restrictions. That is why it's better to se CLOB.

LONG:
- **it is used to hold large amounts of chars.**
- **it has some restrictions:**
  - **a table can have only one column as LONG type.**
  - **we cannot use built-in functions on LONG type.**
- **max size: 2GB**

CLOB:
- **CLOB => Character Large Object**
- **it is used to hold large amounts of chars.**
- **A table can have multiple columns as CLOB type.**
- **we can use built-in functions on CLOB type.**
- **max size: 4 GB**

Example:

Feedback CLOB
Product_Features CLOB

Char(n) Varchar2(n) LONG CLOB	• ASCII Code Char data types • These can hold ENGLISH language chars only • Single Byte Char Data types
nChar(n) nVarchar2(n) nCLOB n => national	• UNI code char data types • These can hold ENGLISH + other LANG chars • Multi Byte char data types

nChar(n)	• it is fixed length data type • it is used to hold fixed length chars • n => no of chars • max size: 2000 Bytes [1000 chars]
nVarchar2(n)	• it is variable length data type • it is used to hold variable length chars • n => no of chars • max size: 4000 Bytes [2000 chars]
nCLOB	• it is used to hold variable length chars • to hold more than 2000 chars we use it

In C:

char ch='A'; // 1 Byte => ASCII

In Java:

char ch='A'; // 2 Bytes => UNI

ASCII:

- ASCII => American Standard Code for Information Interchange
- it is a coding system.
- 256 chars are coded.
- ranges from 0 to 255.
- 255 => 1111 1111 [1 Byte]
- English Lang letters + digits + special chars

UNI:

- UNI => UNIversal
- it is a coding system.
- 65536 chars are coded.
- ranges from 0 to 65535.
- 65535 => 1111 1111 1111 1111 [2 Bytes]
- UNI = ASCII + other language chars
- it is extension of ASCII

Integer related data types:

- integer =>number without decimal places
- Examples: 1234, 78, 17

NUMBER(p):

- it is used to hold integers.
- p => precision => max no of digits
- p valid range is: 1 to 38

Examples:

EMPID NUMBER(4) -9999 TO 9999

1234
1235
1236
786
9
98
9999
10000 ERROR

Max marks: 100

Maths_marks NUMBER(3) -999 TO 999

68
9
123
890
999
1000 ERROR

AGE NUMBER(2)

MOBILE_NUMBER NUMBER(10)

AADHAR_NUMBER NUMBER(12)

CREDIT_CARD_NUMBER NUMBER(16)

Note:
integer and int are alias names of number(38)

integer = int = number(38)

Floating Point related data types:

Number(p,s):
- p => precision => max no of digits
- s => scale => max no of decimal places
- it is used to hold float values.

Example:

-999.99 TO 999.99

AVRG NUMBER(5,2)

67.89
786.34
999.99
1000 => ERROR
123.45678923 => 123.46
123.45378954 => 123.45

Max marks: 100
3 subjects
300/3 = 100.00

max avrg:

100.00

p=5 s=2

max sal: 100000.00

-999999.99 TO 999999.99

SALARY NUMBER(8,2)

25000.00
100000.00
900000.00
1000000.00 ERROR

Height NUMBER(2,1) -9.9 TO 9.9

5.3
5.0
5.9
8.5
9.9
10 ERROR

Date & Time Related data types:

DATE:

- **It is used to hold date values.**
- **it can hold date, month, year, hours, minutes and seconds.**
- **it cannot hold fractional seconds.**
- **by default it will not display time.**
- **Default oracle date format: DD-MON-YY.**
- **default time: 12:00:00 AM [mid night time]**
- **it is fixed length data type.**
- **max size: 7 bytes.**

Examples:
Date_Of_Birth DATE
Date_Of_retirement DATE
Ordered_Date DATE
Delivery_Date DATE

Timestamp:

- **introduced in ORACLE 9i version.**
- **It is used to hold date and time.**
- **it can hold date, month, year, hours, minutes, seconds and fractional seconds.**
- **It is extension of DATE type.**
- **by default it displays time.**
- **default format: DD-MON-YY HH:MI:SS:FF AM**
- **it is fixed length data type.**
- **max size: 11 Bytes**

Examples:
Transaction_date_time TIMESTAMP
login_date_time TIMESTAMP
manufactured_Date_time TIMESTAMP

Differences b/w DATE and TIMESTAMP:

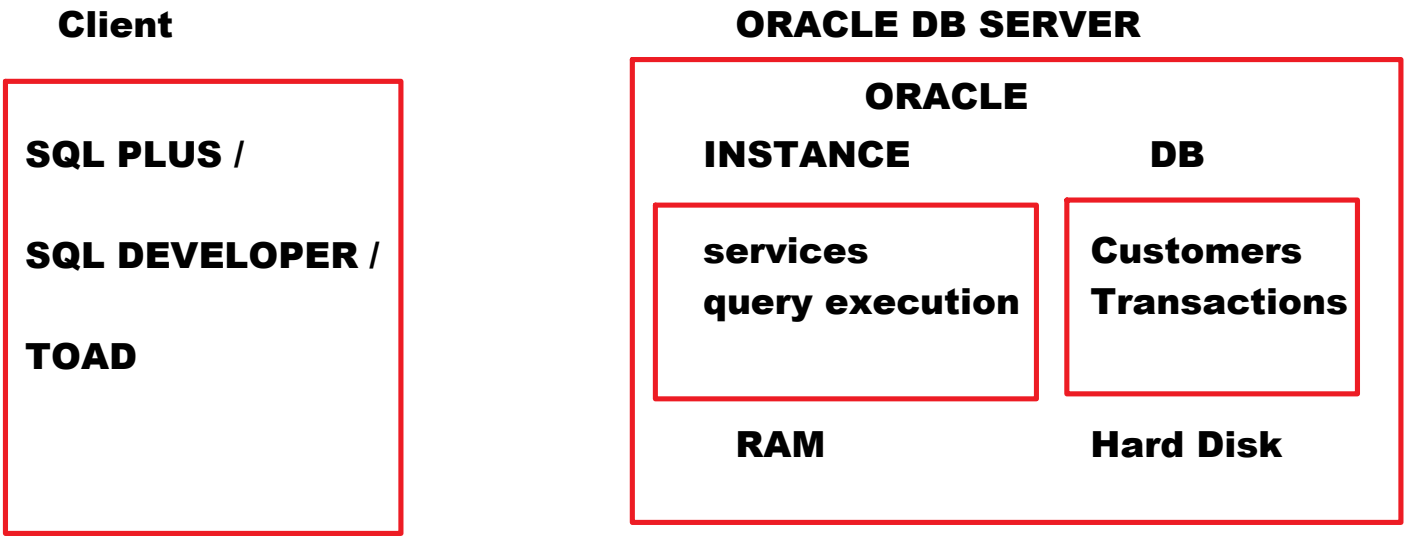
DATE	TIMESTAMP
• it cannot hold fractional seconds. • it does not display time by default • 7 Bytes Example: DOJ DATE	• it can hold fractional seconds • by default it displays time • 11 Bytes Example: Transaction_date_time TIMESTAMP

T1

fixed length	F1 CHAR(10)	F2 VARCHAR2(10)	variable length
10	RAJU6spaces	RAJU	4
10	NARESH4spaces	NARESH	6
10	SAI7spaces	SAI	3

ORACLE 21C Software Link:
bit.ly/oracle21csoftware

Oracle Installation Video Link:
bit.ly/oracle21cinstallation



DB SERVER = INSTANCE + DB

ORACLE:

- **is a server side software.**
- **it is used to maintain the database in the form of tables.**

DB SERVER = INSTANCE + DB

SQL PLUS:

- **Client side software.**
- **Using this software, we can connect and communicate with ORACLE DB.**

Note:
When we install ORACLE software, along with it SQL PLUS software will be installed.

Opening SQL PLUS:

- **Press windows+R. it displays RUN dialog box.**
- **type "sqlplus"**
- **click on OK.**

to login as DBA:

- **username: system**
- **password: naresh [at the time of ORACLE installation you have given password in 4th step]**

SQL> <type queries>

Creating User:

Syntax:

```
CREATE USER <username>
IDENTIFIED BY <password>;
```

Example:

Login as DBA:
username: system

password: naresh

```
CREATE USER c##batch730am
IDENTIFIED BY nareshit;
```

Output:

User created.

```
GRANT connect, resource, unlimited tablespace
TO c##batch730am;
```

Output:

Grant succeeded.

privilege => permission

connect	• is a privilege. • is a permission for login
resource	• is a privilege. • is a permission to create database resources like tables, indexes, procedures, functions, packages, triggers.
create table	• is a privilege • is a permission to create the table
create procedure	• is a privilege • is a permission to create the procedure
unlimited tablespace	• is a privilege • is a permission to insert the records

To see current user name:

SHOW USER

Output:

user is "SYSTEM"

To login from SQL command prompt:

Syntax:

CONN[ECT] <username>/<password>

Example:

CONN c##batch730am/nareshit

Output:

Connected.

create user with the name c##vijay
with the password naresh. give permission to login, for creating table and for inserting records:

CREATE USER c##vijay
IDENTIFIED BY naresh;
Output:
User created.

GRANT connect, resource, unlimited tablespace
TO c##vijay;

Changing password:

Syntax:
ALTER USER <user_name>
IDENTIFIED BY <new_password>;

Modify c##batch730am user's password as naresh:

Login as DBA:
username: system
password: naresh

ALTER USER c##batch730am
IDENTIFIED BY naresh;

Note:
Username is not case sensitive
C##BATCH730AM = c##batch730am = C##BatCH730aM

password is case sensitive

Modifying DBA password:

username: sys as sysdba
password: [don't enter any password]

SQL> ALTER USER system IDENTIFIED BY nareshit;

Dropping User:

Syntax:
DROP USER <username> [CASCADE];

Example:
DROP USER c##vijay CASCADE;

Note:
If user is empty no need to write CASCADE.

If user is not empty we must write CASCADE.

To clear screen:

Syntax:

CL[EAR] SCR[EEN]

Example:

SQL> CLEAR SCREEN

(or)

SQL> CL SCR

Creating tables and Inserting records

Monday, June 17, 2024 7:50 AM

Creating tables and Inserting records:

Syntax to create the table:

```
CREATE TABLE <table_name>
(
<column_name> <data_type> [,
<column_name> <data_type> ,
.
.]
);
```

Syntax to INSERT the records:

```
INSERT INTO <table_name>[(<columns_list>)]
VALUES(<values_list>);
```

Example-1:

STUDENT

SID	SNAME	AVRG
1234	Kiran	56.78
1235	Sai	78.92

AVRG: 100.00

SID	NUMBER(4)
SNAME	VARCHAR2(10)
AVRG	NUMBER(5,2)

Creating table:

```
CREATE TABLE student
(
sid NUMBER(4),
sname VARCHAR2(10),
avrg NUMBER(5,2)
);
```

Output:
Table created.

Inserting records:

1234	Kiran	56.78
1235	Sai	78.92

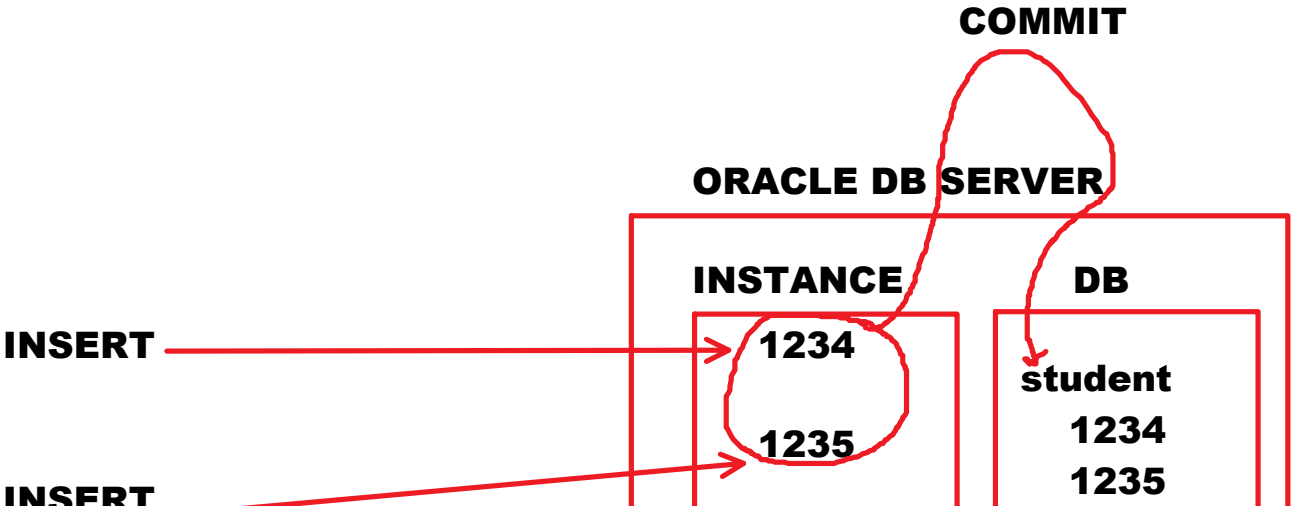
```
INSERT INTO student VALUES(1234,'KIRAN',56.78);
```

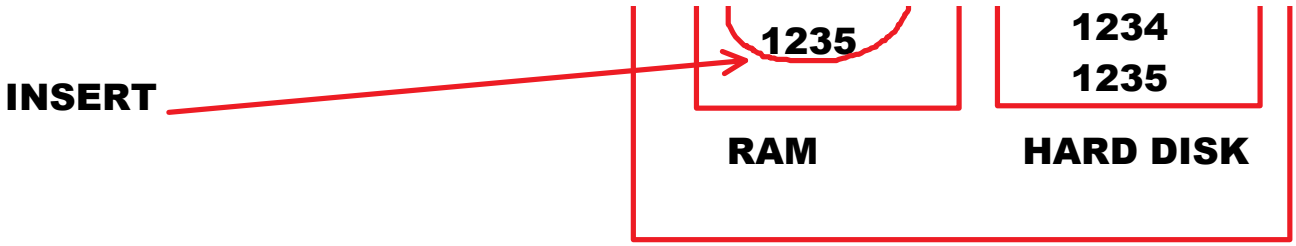
Output:
1 row created.

```
INSERT INTO student VALUES(1235,'SAI',78.92);
```

Output:
1 row created.

```
COMMIT;      --it saves the data. it moves data from INSTANCE to DB
```





Note:
SQL is not case sensitive.
CREATE = create = CReaTe => all are same

Every SQL command ends with ; [semicolon]

Inserting multiple records using parameters:

```
INSERT INTO student VALUES(&sid, '&sname', &avrg);
Output:
Enter value for sid: 2002
Enter value for sname: B
Enter value for avrg: 45.23
INSERT INTO student VALUES(&sid, '&sname', &avrg)
INSERT INTO student VALUES(2002, 'B', 45.23)
1 row created.

/
Output:
Enter value for sid: 2003
Enter value for sname: C
Enter value for avrg: 52.89

/
Output:
Enter value for sid: 2004
```

Enter value for sname: D
Enter value for avrg: 88.99

Note:

- **Parameter Concept is used to read the values at run time.**

  Syntax:
  &<text>

  Example:
  &sid

  Output:
  Enter value for sid:

- **/ = RUN = R**
  It runs recent query in memory. It means, it runs above query

Inserting limited column values:

STUDENT

SID	SNAME	AVRG
5001	ABC	

INSERT INTO student VALUES(5001,'ABC');
Output:
ERROR: not enough values

INSERT INTO student(sid, sname) VALUES(5001,'ABC');

Output:
1 row created.

Example-2:

EMPLOYEE

EMPID	ENAME	GENDER	SAL	DOJ
1001	AA	M	12000	25-DEC-23
1002	BB	F	15000	17-AUG-20

100000.00

empid	NUMBER(4)
ename	VARCHAR2(10)
gender	CHAR(1)
sal	NUMBER(8,2)
doj	DATE

```
CREATE TABLE employee
(
empid NUMBER(4),
ename VARCHAR2(10),
gender CHAR(1),
sal NUMBER(8,2),
doj DATE
);
```

Output:
Table created.

to see table structure:

Syntax:

DESC[RIBE] <table_name>

Example:

DESC employee

Output:

```
NAME                    TYPE
-----------------------------------------
EMPID                   NUMBER(4)
ENAME                   VARCHAR2(10)
..                       ..
```

TO see all tables list which are created by user:

User_Tables:

- **it is a system table / built-in table / readymade table.**
- **It maintains all tables information which are created by a user.**

DESC user_tables

Output:

```
NAME
----------------
TABLE_NAME
..
..
```

SELECT table_name FROM user_tables;

Output:

```
TABLE_NAME
----------------------
STUDENT
EMPLOYEE
```

Inserting Records:

1001	AA	M	12000	25-DEC-23
1002	BB	F	15000	17-AUG-20

```
INSERt INTO employee
VALUES(1001, 'AA', 'M', 12000, '25-DEC-2023');
```

string

```
DOJ    DATE
--------------------
25-DEC-23          date
```

Implicit Conversion

Note:

- To_Date() function is used to convert string to date.

1002	BB	F	15000	17-AUG-20

```
INSERT INTO employee
VALUES(1002,'BB','F',15000,to_date('17-AUG-2020'));
```

string

```
DOJ    DATE
---------------------
17-AUG-20          date
```

to_date()
explicit conversion

insert emp record with today's date:

1003	C	M	20000	today's date => 18-JUN-24

```
INSERT INTO employee
VALUES(1003,'C','M',20000,sysdate);
```

Output:
1 row created.

Note:
sysdate:

- is a built-in function.
- it returns current system date.

```
COMMIT;     --it saves data
```

to see table data:

```
SELECT * FROM employee;
```

1004	D	null	18000	null

```
INSERT INTO employee VALUES(1004,'D',18000);
```

Output:
ERROR: not enough values

```
INSERT INTO employee(empid, ename, sal)
VALUES(1004,'D',18000);
```

NOTE:
For every data type default value is NULL.

inserting limited column values by changing order of columns:

1005			13000	

```
INSERT INTO employee(sal, empid )
VALUES(13000, 1005);

COMMIT;

SELECT * FROM employee;
```

Example-3:

EMPLOYEE1

EMPID	ENAME	LOGIN_DATE_TIME
5001	A	18-JUN-24 10:30.0.0 AM
5002	B	18-JUN-24 2:30.0.0 PM
5003	C	current sys date and time

```
CREATE TABLE employee1
(
empid NUMBER(4),
ename VARCHAR2(10),
login_date_time TIMESTAMP
);
```

5001	A	18-JUN-24 10:30.0.0 AM

```
INSERT INTO employee1
VALUES(5001,'A','18-JUN-2024 10:30 AM');
```

Output:

ERROR

```
INSERT INTO employee1
VALUES(5001,'A','18-JUN-2024 10:30');
```

string

Output:

implicit conversion

```
LOGIN_DATE_TIME  TIMESTAMP
-----------------------------
18-JUN-24  10:30:0.0 AM   TIMESTAMP
```

Note:
default time: 12:00:00.00 AM

5002	B	18-JUN-24 2:30.0.0 PM

```
INSERT INTO employee1
VALUES(5002,'B',to_timestamp('18-JUN-2024 2:30:0.0 PM'));
```

string

explicit conversion

```
LOGIN_DATE_TIME  TIMESTAMP
-----------------------------------------------
18-JUN-24 2:30:0.0 PM        timestamp
```

5003	C	current sys date and time

```
INSERT INTO employee1
VALUES(5003,'C',systimestamp);

COMMIT;

SELECT * FROM employee1;
```

Types of Conversions:

2 Types:

- implicit conversion
- explicit conversion

implicit conversion:
If conversion is done implicitly by ORACLE then it is called "Implicit Conversion".

Example:

```
SELECT '100' + '200' FROM dual;
       string  string
```

Output: **implicit conversion**
300

```
num   num
100 + 200
300 => num
```

Explicit Conversion:

- **if conversion is done using built-in function then it is called "Explicit Conversion".**

Example:

```
SELECT to_number('100') FROM dual;
```

string

↓

num

100

to_number()

explicit conversion

Note:

Implicit Conversion degrades the performance.

that is why always do explicit conversion. For explicit conversion we use built-in functions like to_number(), to_date(), to_char().

Setting Pagesize and Linesize

Wednesday, June 19, 2024 8:58 AM

Setting Pagesize and Linesize:

SQL> SHOW ALL

Output:
LINESIZE 80
PAGESIZE 14

Note:
default page size is 14
default line size is 80

Setting pagesize:

Syntax:
SET PAGES[IZE] <value>

Example:
SQL> SET PAGES 200

Setting linesize:

Syntax:
SET LINES[IZE] <value>

Example:
SQL> SET LINES 200

SQL> SET PAGES 200
SQL> SET LINES 200 **(or)** **SQL> SET PAGES 200 LINES 200**

It will be applicable for 1 session

COLUMN ALIAS

Thursday, June 20, 2024 8:24 AM

COLUMN ALIAS:

- we can give temporary name to the column. this is called column alias.
- alias => another name / alternative name
- To change the column headings in output we use column alias.
- AS keyword is used to give column alias.
- Using AS keyword is optional.
- To give column alias in multiple words or to maintain the case we must enclose column alias in double quotes.

Example:

```
SELECT ename, sal FROM emp
Output:
    ENAME        SAL
    -------------------------------

SELECT ename AS A, sal AS B
FROM emp;

(or)

SELECT ename A, sal B
FROM emp;


Output:
    A                B
    -------------------------------
```

DRL

Wednesday, June 19, 2024 8:18 AM

DRL / DQL:

- **DRL => Data Retrieval Language**
- **DQL => Data Query Language**

- **Retrieval => opening existing data**

  checking balance
  searching for products
  transaction statement

ORACLE SQL provides one DRL command. i.e:

- **SELECT**

SELECT:

- **It us used to retrieve (fetch / select) the data from table.**

- **Using SELECT command we can select:**
  - **All columns and all rows**
  - **All columns and specific rows**
  - **Specific columns and all rows**
  - **Specific columns and specific rows**

  Syntax:

```
SELECT <columns_list>
FROM <tables_list>
WHERE <condition>;
```

SQL	ENGLISH
QUERIES	SENTENCES
CLAUSES	WORDS

SELECT clause:

- **it is used to specify columns list**

  Syntax:

  SELECT <columns_list>

  Example:

  SELECT sname, avrg

FROM clause:

- **it is used to specify tables list**

  Syntax:

  FROM <tables_list>

  Examples:

  FROM student

  FROM student, marks

WHERE clause:

- **it is used to specify filter condition.**
- **it filters the rows.**
- **WHERE condition will be applied on every row.**

  Syntax:

  WHERE <condition>

  Example:

  WHERE sid=1234

  WHERE avrg>=60

All columns and all rows:

Display all columns and all rows of emp table:

```
SELECT * FROM emp;
```

*	All Columns

Note:
SELECT * FROM emp

*	empno,ename,job,mgr,hiredate,sal,comm,deptno

Above query will be rewritten by ORACLE as following:

```
SELECT empno,ename,job,mgr,hiredate,sal,comm,deptno
FROM emp
```

All columns and specific rows:

Display the emp records whose salary is 3000:

```
SELECT *
FROM emp
WHERE sal=3000;
```

Execution Order:
FROM
WHERE
SELECT

FROM emp

EMP

EMPNO	ENAME	SAL
1001	A	2500
1002	B	3000
1003	C	5000
1004	D	3000

WHERE sal=3000

2500=3000	F
3000=3000	T
5000=3000	F
3000=3000	T

EMPNO	ENAME	SAL
1002	B	3000
1004	D	3000

SELECT *
*** = empno, ename, sal**

EMPNO	ENAME	SAL
1002	B	3000
1004	D	3000

Specific columns and all rows:

Display all emp names and salaries:

```
SELECT ename, sal
FROM emp;
```

Specific columns and specific rows:

Display emp names and salaries whose salary is 3000:

```
SELECT ename, sal
FROM emp
WHERE sal=3000;
```

All Columns	SELECT *
All Rows	don't write WHERE
Specific Columns	SELECT ename, sal
Specific Rows	WHERE sal=3000

Operators in ORACLE SQL:

- **OPERATOR is a symbol that is used to perform operations like arithmetic or logical operations.**
- **ORACLE SQL provides following Operators:**

Arithmetic	**+ - * /**
Relational / Comparison	**> < >= <= = equals != / <> / ^= not equals**
Logical	**AND OR NOT**
Special / Comparison	**IN** NOT IN **BETWEEN AND** NOT BETWEEN AND **LIKE** NOT LIKE **IS NULL** IS NOT NULL **ALL** **ANY** **EXISTS**
Set	**UNION** **UNION ALL** **INTERSECT** **MINUS**
Concatenation	**||**

Arithmetic Operators:
Arithmetic operators are used to perform arithmetic operations.

ORACLE SQL provides following Arithmetic Operators:

+	Addition
-	Subtraction
*	Multiplication
/	Divison

In C or Java:
5/2 = 2
int/int = int
5%2 = 1

In SQL:
5/2 = 2.5
MOD(5,2) => 1

Examples on Arithmetic Operators:

Calculate Annual salary of all emps:

```
SELECT ename, sal, sal*12 FROM emp;
Output:
   ENAME      SAL        SAL*12
   ----------------------------------------------
```

```
SELECT ename, sal, sal*12 AS annual_sal
FROM emp;
Output:
   ENAME      SAL        ANNUAL_SAL
   ------------------------------------------------------
```

```
SELECT ename, sal, sal*12 AS annual salary
FROM emp;
Output:
ERROR
```

```
SELECT ename, sal, sal*12 AS "annual salary"
FROM emp;
Output:
```

ENAME SAL annual salary

Calculate experience of all emps:

```
SELECT ename, hiredate,
TRUNC((sysdate-hiredate)/365) AS experience
FROM emp;
```

Calculate TA, HRA, TAX and GROSS SALARY:
10% on sal => TA
20% on sal => HRA
2% on sal => TAX
GROSS = bsal + TA + HRA - TAX

```
SELECT ename, sal,
sal*0.1 AS TA,
sal*0.2 AS HRA,
sal*0.02 AS TAX,
sal+sal*0.1+sal*0.2-sal*0.02 AS GROSS
FROM emp;
```

Assignment:

STUDENT

SID	SNAME	M1	M2	M3
1001	A	66	78	46
1002	B	78	34	92

m1+m2+m3	(m1+m2+m3)/3
66+78+46 = 190	190/3
78+34+92 = 204	204/3

calculate total marks and avrg marks

Relational Operators / Comparison Operators:

- **Relational operator is used to compare column value with 1 value.**

  Syntax:

  <column> <relational_operator> <value>

  Examples:

  sal=3000
  sal>3000
  sal<3000
  sal>=3000
  sal<=3000
  sal!=3000

Display all managers records:

```
SELECT ename, job, sal
FROM emp
WHERE job='manager';
```

MANAGER = manager FALSE

Output:

no rows selected

Note:

- **SQL is not case sensitive language. But, string comparison is case sensitive.**

```
SELECT ename, job, sal
FROM emp
WHERE job='MANAGER';
```

Output:

displays all managers records

Display the emp records who are working in deptno 20:

```
SELECT ename, sal, deptno
FROM emp
WHERE deptno=20;
```

Display the emp records whose salary is more than 2500:

```
SELECT ename, sal
FROM emp
WHERE sal>2500;
```

Display the emp records whose salary is 3000 or more:

```
SELECT ename, sal
FROM emp
WHERE sal>=3000;
```

Display the emp records whose salary is 1200 or less:

```
SELECT ename, sal
FROM emp
WHERE sal<=1200;
```

Note:

CALENDAR order is ASCENDING ORDER [small to big]

After 2023

1-JAN-23
2-JAN-23
3-JAN-23
.
.

31-DEC-23
1-JAN-24 > '31-DEC-23'
2-JAN-24
.
.
31-DEC-24

Display the emp records who joined after 1981:

1-JAN-1981
2-JAN-1981
.
.
31-DEC-1981
1-JAN-1982 > '31-DEC-1981'
.
.
31-DEC-1982
.
.

```
SELECT ename, hiredate
FROM emp
WHERE hiredate>'31-DEC-1981';
```

Display the emp records who joined before 1981:

.
.
31-DEC-1980 < '1-JAN-1981'
1-JAN-1981

```
SELECT ename, hiredate
FROM emp
WHERE hiredate<'1-JAN-1981';
```

Display all emp records except managers:

```
SELECT ename, job, sal
FROM emp
WHERE job!='MANAGER';
```

```
CLERK!=MANAGER   T
MANAGER!=MANAGER  F
ANALYST!=MANAGER  T
```

Display all emp records except deptno 30 emps:

```
SELECT ename, sal, deptno
FROM emp
WHERE deptno!=30;
```

Logical Operators:

- **Logical Operators are used to perform logical operations like logical AND, logical OR, logical NOT.**

ORACLE SQL provides following Logical Operators:

- **AND**
- **OR**
- **NOT**

AND:

- **it is used to perform logical AND operations.**
- **it is used to separate multiple conditions.**
- **If all conditions are satisfied then that whole condition is TRUE.**

Syntax:

```
<condition1> AND <condition2>
```

OR:

- **it is used to perform logical OR operations.**
- **it is used to separate multiple conditions.**
- **If any one condition is satisfied then that whole condition is TRUE.**

Syntax:

<condition1> OR <condition2>

Note:

AND	**All conditions should be satisfied**
OR	**At least one condition should be satisfied**

Truth Table:

c1 => condition1
c2 => condition2

c1	c2	c1 AND c2	c1 OR c2
T	T	T	T
T	F	F	T
F	T	F	T
F	F	F	F

Examples on AND OR:

Display all managers and clerks records:

```
SELECT ename, job, sal
FROM emp
WHERE job='MANAGER' OR  job='CLERK';
```

Display the emp records who are working in deptno 10 and 30:

```
SELECT ename, sal, deptno
FROM emp
WHERE deptno=10 OR deptno=30;
```

Display the emp records whose salary is 2450 or more and 3000 or less [whose sal b/w 2450 and 3000]:

```
SELECT ename, sal
FROM emp
WHERE sal>=2450 AND sal<=3000;

SAL
---------
5000
2500
1000
3000
2450
```

Display the emp records who joined in 1981:

```
SELECT ename, hiredate
FROM emp
WHERE hiredate>='1-Jan-1981' AND hiredate<='31-DEC-1981'

1-JAN-1981
2-JAN-1981
.
.
30-DEC-1981
31-DEC-1981
```

Display the emp records whose empnos are: 7499, 7698, 7900:

```
SELECT empno, ename, sal
FROM emp
WHERE empno=7499 OR empno=7698 OR empno=7900;
```

Display the emp records whose names are: ALLEN, MILLER, SCOTT:

```
SELECT ename, sal
FROM emp
WHERE ename='ALLEN' OR ename='MILLER' OR ename='SCOTT';
```

Display the managers records whose salary is more than 2900:

```
SELECT ename, job, sal
FROM emp
WHERE job='MANAGER' AND sal>2900;
```

Display the managers records who joined after april 1981:

```
SELECT ename, job, hiredate
FROM emp
WHERE job='MANAGER' AND hiredate>'30-APR-1981';
```

Display all managers and clerks records whose salary is more than 2800:

```
SELECT ename, job, sal
FROM emp
WHERE (job='MANAGER' OR job='CLERK') AND sal>2800;
```

Example:

STUDENT

SID	SNAME	M1	M2	M3
1001	A	70	90	80
1002	B	50	30	60

```
CREATE TABLE student
(
sid NUMBER(4),
sname VARCHAR2(10),
m1 NUMBER(3),
m2 NUMBER(3),
m3 NUMBER(3)
);

INSERT INTO student VALUES(1001,'A',70,90,80);
INSERT INTO student VALUES(1002,'B',50,30,60);
COMMIT;
```

display passed students records:
max marks: 100
min marks: 40

```
SELECT * FROM student
WHERE m1>=40 AND m2>=40 AND m3>=40;
```

display failed students records:

```
SELECT * FROM student
WHERE m1<40 OR m2<40 OR m3<40;
```

NOT:

- **it is used to perform logical NOT operations.**

Truth table:

condn	NOT(condn)
T	NOT(T) => F
F	NOT(F) => T

Display all emp records except managers:

```
SELECT ename, job, sal
FROM emp
WHERE NOT(job='MANAGER');
```

```
JOB             NOT(job='MANAGER')
-------------   ------------------------------
MANAGER         MANAGER = MANAGER  => NOT(T) => F
CLERK           CLERK = MANAGER => NOT(F) => T
ANALYST         ANALYST = MANAGER => NOT(F) => T
```

Special Operators / Comparison Operators:

IN:

- **It is used to compare column value with a list of values.**
- **It avoids of writing multi equality conditions using OR.**

Syntax:

```
<column> IN(<values_list>)
```

Example:

```
sal IN(3000,800)
```

if column value is in LIST, condition is TRUE

if column value is not in LIST, condition is FALSE

Examples on IN:

Display the emp records whose salary is 3000 or 800:

```
SELECT ename, sal
FROM emp
WHERE sal=3000 OR sal=800;
```

(or)

```
SELECT ename, sal
FROM emp
WHERE sal IN(3000, 800);
```

sal=3000 OR sal=800	sal IN(3000, 800)

```
sal
----------
2500    not in list  F
800     is in list  T
4000    not in list  F
3000    is in list    T
```

Display all managers and clerks records:

```
SELECT ename, job, sal
FROM emp
WHERE job IN('MANAGER', 'CLERK');
```

```
JOB
----------
MANAGER     T
```

```
ANALYST     F
CLERK       T
SALESMAN    F
```

Display the emp records whose empnos are: 7499, 7698, 7900:

```
SELECT *
FROM emp
WHERE empno IN(7499,7698,7900);

   EMPNO
   ------------
   7698     T
   7800     F
```

Display the emp records whose names are: ALLEN, SCOTT, ADAMS

```
SELECT ename, sal
FROM emp
WHERE ename IN('ALLEN','SCOTT','ADAMS');
```

Display the emp records who are working in deptno 10 and 30:

```
SELECT ename, sal, deptno
FROM emp
WHERE deptno IN(10,30);
```

Display all emp records except deptno 10 and 30 emps:

```
SELECT ename, sal, deptno
FROM emp
WHERE deptno NOT IN(10,30);
```

if deptno is not in LIST, condition is TRUE
if deptno is in LIST, condition is FALSE

```
deptno          deptno NOT IN(10,30)
----------------  ---------------------------------
10                10  F
20                20  T
30                30  F
40                40  T
```

Display all emp records except managers and clerks:

```
SELECT ename, job, sal
FROM emp
WHERE job NOT IN('MANAGER', 'CLERK');
```

```
ANALYST  T
MANAGER F
```

BETWEEN AND:

- It is used to compare column value with range of values.

Syntax:

<column> BETWEEN <lower> AND <upper>

Example:

sal BETWEEN 2450 AND 3000

If column value falls under the range, condition is TRUE
If column value not falls under the range, condition is FALSE

Examples on BETWEEN AND:

Display the emp records whose salary is 2450 or more and 3000 or less [whose sal b/w 2450 and 3000]:

```
SELECT ename, sal
FROM emp
WHERE sal>=2450 AND sal<=3000;

(or)

SELECT ename, sal
FROM emp
WHERE sal BETWEEN 2450 AND 3000;

SAL
------------
1000    F
2500    T
4000    F
2450    T
6000    F
3000    T
```

Display the emp records who joined in 1982:

```
1-JAN-1982
.
.
31-DEC-1982

SELECT ename, hiredate
FROM emp
WHERE hiredate BETWEEN '1-JAN-1982' AND '31-DEC-1982';

 HIREDATE
 --------------------
```

25-DEC-1980 F
17-AUG-1982 T

Display the emp records whose empnos are between 7600 to 7800:

```
SELECT *
FROM emp
WHERE empno BETWEEN 7600 AND 7800;
```

Display the emp records whose salary is less than 1000 or more than 3000 [whose sal not between 1000 and 3000]:

```
SELECT ename, sal
FROM emp
WHERE sal<1000 OR sal>3000;
```

(or)

```
SELECT ename, sal
FROM emp
WHERE sal NOT BETWEEN 1000 AND 3000;
```

if sal is not between 1000 and 3000 then condition is TRUE

```
SAL
----------
5000 T
2500 F
```

```
SELECT ename, sal
FROM emp
WHERE sal BETWEEN 3000 AND 2450;
```

What is the output?

A. gives sal b/w 2450 and 3000
B. ERROR
C. NO ROWS SELECTED
D. NONE

Answer: C

LIKE:

- It is used to compare column value with text pattern.

Syntax:

```
<column_name> LIKE <text_pattern>
```

ORACLE SQL provides 2 wildcard characters to specify search pattern. They are:

%	replaces 0 or any no of chars
_	replaces 1 char

Examples on LIKE operator:

Display the emp records whose names are started with S:

```
SELECT *
FROM emp
WHERE ename LIKE 'S%';
```

Display the emp records whose names are ended with S:

```
SELECT *
FROM emp
WHERE ename LIKE '%S';
```

Display the emp records whose names are started with A:

```
SELECT *
FROM emp
WHERE ename LIKE 'A%';
```

Display the emp names whose names are ended with RD:

```
SELECT *
FROM emp
WHERE ename LIKE '%RD';
```

Display the emp records whose names are started and ended with S:

```
SELECT *
FROM emp
WHERE ename LIKE 'S%S';
```

Display the emp names whose names are having 4 letters:

```
SELECT *
FROM emp
WHERE ename LIKE '____';
```

Display the emp records whose name's 2nd char is A:

```
SELECT *
```

```
FROM emp
WHERE ename LIKE '_A%';
```

Display the emp records whose names are having A letter:

```
SELECT *
FROM emp
WHERE ename LIKE '%A%';
```

Display the emp records who joined in DECEMBER month:

```
SELECT ename, hiredate
FROM emp
WHERE hiredate LIKE '%DEC%';
```

Display the emp records whose are getting 3 digit salary:

```
SELECT ename, sal
FROM emp
WHERE sal LIKE '___';
```

Display the emp records whose names are not started with S:

```
SELECT *
FROM emp
WHERE ename NOT LIKE 'S%';
```

IS NULL:

- **It is used for null comparison.**

Syntax:

```
<column> IS NULL
```

EXAMPLE:

SAL IS NULL

```
SAL
------------
6000      F
null      T
7000      F
null      T
```

Display the emp records who are not getting commission [whose comm is null]:

```
SELECT ename, sal, comm
FROM emp
WHERE comm=null;
Output:
no rows selected
```

NULL=NULL FALSE

Note:
for null comparison we cannot use =
we must use IS NULL

```
SELECT ename, sal, comm
FROM emp
WHERE comm IS null;
```

Display the emp records who are getting commission:

```
SELECT ename, sal, comm
FROM emp
WHERE comm IS NOT NULL;
```

Concatenation Operator:

- **Symbol: ||**
- **Concatenation => Combining**
- **It is used to combine 2 strings.**

Syntax:

```
<string1> || <string2>
```

Example:

```
SELECT 'RAJ' || 'KUMAR' FROM dual;
Output:
RAJKUMAR
```

Display output as following:

```
SMITH works as CLERK
ALLEN works as SALESMAN
BLAKE works as MANAGER

SELECT ename || ' works as ' || job
FROM emp;
```

Display output as following:

```
SMITH joined on 17-DEC-80

SELECT ename || ' joined on ' || hiredate
FROM emp;
```

Display output as following:

```
SMITH works as CLERK and earns 800

SELECT ename || ' works as ' || job || ' and earns ' || sal
FROM emp;
```

Assignment:

CUSTOMER

CID	FNAME	MNAME	LNAME
1234	RAJ	KUMAR	VARMA
1235	..	..	..

Combine first name, middlename and last name:

fname || ' ' || mname || ' ' || lname

RAJ KUMAR VARMA

NULL

Monday, June 24, 2024 8:21 AM

NULL:

- **NULL means empty / blank / no value.**

- **When we don't know the value or when we are unable to insert the value we insert NULL.**

- **NULL is not equals to 0 or space.**

- **If null is participated in operation then result will be NULL**

  Example:

  SELECT 100+200+null FROM dual;

  Output:

  null

- **For null comparison we cannot use =.**
  We must use IS NULL.

Note:

we can insert NULL in 2 ways. they are:

- **Direct way: use NULL keyword**
- **Indirect way: insert limited column values**

Example:

EMP21

EMPID	ENAME	SAL

```
CREATE TABLE emp21
(
empid NUMBER(4),
ename VARCHAR2(10),
sal NUMBER(8,2)
);
```

1001	A	

1st way: direct way => using null keyword:

```
INSERT INTO emp21 VALUES(1001,'A',null);
```

1002	B	

2nd way: indirect way => insert limited column values:

```
INSERT INTO emp21(empid,ename) VALUES(1002,'B');
```

1003		7000

```
INSERT INTO emp21 VALUES(1003,'',7000);
(or)
INSERT INTO emp21 VALUES(1003,NULL,7000);
```

EMPLOYEE

EMPID	ENAME	SAL
1001	A	12000
1002	B	10000
1003	C	

NULL

when we don't know the value we insert NULL

STUDENT

SID	SNAME	M1 NUMBER(3)
1001	A	76
1002	B	0
1003	C	66
1004	D	

NULL

when we unable to insert value we insert NULL

unable to insert ABSENT

UPDATE

Tuesday, June 25, 2024 8:04 AM

UPDATE:

- **UPDATE command is used to modify table data.**

- **Using UPDATE command we can modify:**
  - **single value of single record**
  - **multiple values of single record**
  - **specific group of records**
  - **all records**

Syntax:

```
UPDATE <table_name>
SET <column_name> = <new_value> [, <column_name> = <new_value>, ..]
[WHERE <condition>];
```

SQL	ENGLISH
QUERIES	SENTENCES
CLAUSES	WORDS

modifying single value of single record:

Increase 2000 rupees salary to the employee whose empno is 7521:

```
UPDATE emp
SET sal=sal+2000
WHERE empno=7521;
```

modifying multiple values of single record:

set job as manager sal as 6000 to an employee whose empno is 7369:

```
UPDATE emp
SET job='MANAGER', sal=6000
WHERE empno=7369;
```

modifying specific group of records:

Increase 20% on salary to all managers:

```
UPDATE emp
SET sal=sal+sal*0.2
WHERE job='MANAGER';
```

modifying all records:

Increase 1000 rupees salary to all emps:

```
UPDATE emp
SET SAL=SAL+1000;
```

Updating records using parameters:

7499 => 10% on sal
7698 => 20% on sal
7900 => 15% on sal

```
UPDATE emp
SET sal=sal+sal*&per/100
WHERE empno=&empno;
Output:
Enter value for per: 10
Enter value for empno: 7499

/
Output:
Enter value for per: 20
Enter value for empno: 7698
```

/

Output:

Enter value for per: 15

Enter value for empno: 7900

Transfer all deptno 10 emps to deptno 20:

```
UPDATE emp
SETdeptno=20
WHERE deptno=10;
```

Increase 10% on sal, 20% on comm to the emps who are getting commission:

```
UPDATE emp
SET sal=sal+sal*0.1, comm=comm+comm*0.2
WHERE comm is not null;
```

Set comm as 900 to the emps who are not getting commission:

```
UPDATE emp
SET comm=900
WHERE comm IS null;
```

Set comm as null to the emps whose empnos are 7369, 7698, 7788:

```
UPDATE emp
SET comm=null
WHERE empno IN(7369,7698,7788);
```

Note:
For null comparison we use IS NULL
For null assignment we use =

Increase 10% on sal to the emps whose annual salary is more than 30000:

```
UPDATE emp
SET sal=sal+sal*0.1
WHERE sal*12>30000;
```

Increase 15% on sal to the emps who are having more than 42years experience:

```
UPDATE emp
SET sal=sal+sal*0.15
WHERE TRUNC((sysdate-hiredate)/365)>42;
```

Example:

EMPLOYEE11

EMPID	ENAME	SAL	TA	HRA	TAX	GROSS
1234	A	18000				
1235	B	20000				

calculate TA, HRA, TAX and GROSS SALARY
10% on sal as TA
20% on sal as HRA
5% on sal as TAX
GROSS = sal+ta+hra-tax

```
CREATE TABLE employee11
(
empid NUMBER(4),
ename VARCHAR2(10),
sal NUMBER(8,2),
ta NUMBER(8,2),
hra NUMBER(8,2),
tax NUMBER(8,2),
gross NUMBER(8,2)
);
```

1234	A	18000				
1235	B	20000				

```
INSERT INTO employee11(empid, ename, sal)
VALUES(1234,'A',18000);

INSERT INTO employee11(empid, ename, sal)
VALUES(1235,'B',20000);

COMMIT;

UPDATE employee11
SET ta=sal*0.1, hra=sal*0.2, tax=sal*0.05;

UPDATE employee11
SET gross=sal+ta+hra-tax;

COMMIT;
```

DELETE

Wednesday, June 26, 2024 8:06 AM

DELETE:

- **DELETE command is used to delete the records from table.**

- **Using DELETE command we can delete:**
  - **single record**
  - **specific group of records**
  - **all records**

Note:
After performing DML operation, to save it use COMMIT.
to cancel it use ROLLBACK.

Syntax:

```
DELETE [FROM] <table_name>
[WHERE <condition>];
```

Examples on delete:

deleting single record:

delete an emp record whose empno is 7788:

```
DELETE FROM emp
```

```
WHERE empno=7788;
```

Output:

1 row deleted.

deleting specific group of records:

delete the emp records who are working in deptno 30:

```
DELETE FROM emp
WHERE deptno=30;
```

deleting all records:

delete all emp records:

```
DELETE FROM emp;
(or)
DELETE emp;
```

delete all managers and clerks records:

```
DELETE FROM emp
WHERE job IN('MANAGER', 'CLERK');
```

delete all deptno 10 and 30 emps records:

```
DELETE FROM emp
WHERE deptno IN(10,30);
```

delete the emp records who are having more than 42 years experience:

```
DELETE FROM emp
WHERE TRUNC((sysdate-hiredate)/365)>42;
```

TCL

Wednesday, June 26, 2024 8:21 AM

TCL:

- **TCL => Transaction Control Language**
- **It deals with transactions.**

- **Transaction:**
  Transaction is a series of actions [SQL commands].
  Examples:
  deposit, withdraw, placing order, fund transfer

  Note:
  Transaction must be successfully finished or cancelled.

  Example:

ACCOUNTS

ACNO	NAME	BALANCE
1234	A	70000-10000 = 60000
1235	B	30000+10000 = 40000

Fund transfer => transaction
transfer 100000= amount from 1234 account to 1235 account:

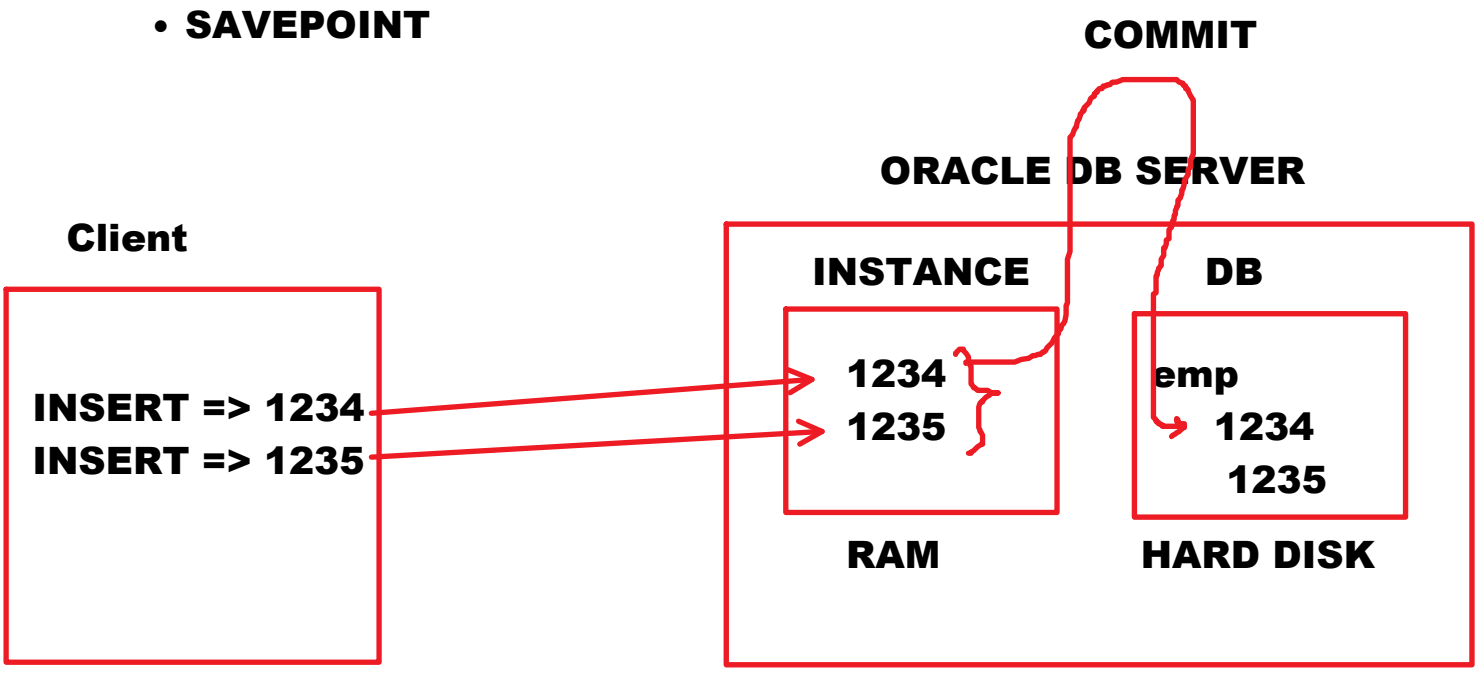
sufficient funds? => SELECT
UPDATE from account balance => UPDATE
UPDATE to account balance => UPDATE

ORACLE SQL provides following TCL commands:

- **COMMIT**
- **ROLLBACK**
- **SAVEPOINT**

COMMIT

- SAVEPOINT



COMMIT:

- it is used to save the transaction.
- when COMMIT command is executed the changes in INSTANCE [RAM] will be moved to DB [HARD DISK].
- COMMIT command makes the changes permanent.

Syntax:

```
COMMIT;
```

ROLLBACK:

- It is used to cancel the transaction.
- When ROLLBACK command is executed, it cancels all uncommitted actions.

Syntax:

```
ROLLBACK [TO <savepoint_name>];
```

Example:

EMPLOYEE12

EMPID	ENAME	SAL

```
CREATE TABLE employee12
(
empid NUMBER(4),
ename VARCHAr2(10),
sal NUMBER(8,2)
);

INSERT INTO employee12 VALUES(1234,'A',12000);
INSERT INTO employee12 VALUES(1235,'B',15000);
COMMIT;

INSERT INTO employee12 VALUES(1236,'C',10000);
SELECT * FROM employee12;
Output:
1234
1235
1236

ROLLBACK;

SELECT * FROM employee12;
Output:
1234
1235
```

SAVEPOINT:

- **It is used to set margin for ROLLBACK.**
- **savepoint names are temporary.**

- **When transaction is ended names will be cleared.**

Syntax:

```
SAVEPOINT <savepoint_name>;
```

Example:

```
CREATE TABLE t1(f1 INT);

BEGIN TRANSACTION                    7.00 AM

INSERT INTO t1 VALUES(1);
INSERT INTO t1 VALUES(2);

SAVEPOINT p1;                        7.10 AM

INSERT INTO t1 VALUES(3);
INSERT INTO t1 VALUES(4);

SAVEPOINT p2;                        7.20 AM

INSERT INTO t1 VALUES(5);
INSERT INTO t1 VALUES(6);

ROLLBACK TO p2;                      7:30 AM
```

to save the transaction	COMMIT
to cancel the transaction	ROLLBACK
to set specific point for ROLBACK	SAVEPOINT

ALTER

Thursday, June 27, 2024 8:11 AM

EMPLOYEE

EMPID	ENAME	SAL
1234	A	6000
1235	B	8000

Structure [columns]
+
data [rows]

Note:
To modify table data we use UPDATE
To modify table structure we use ALTER

ALTER:

- **ALTER => Change**
- **It is used to change table structure.**
- **using ALTER we can:**
  - **Add the columns => ADD**
  - **Rename the columns => RENAME COLUMN**
  - **Drop the columns => DROP**
  - **Modify the field sizes => MODIFY**
  - **Modify the data types => MODIFY**

Syntax:

```
ALTER TABLE <name> [ADD(<field_definitions>)]
                   [RENAME COLUMN <old_name> TO <new_name>]
                   [DROP COLUMN <column_name>]
                   [DROP(<columns_list>)]
                   [MODIFY(<new_field_definitions>)];
```

Example on ALTER:

STUDENT

SID	SNAME

```
CREATE TABLE student
(
sid NUMBER(4),
sname VARCHAR2(10)
);
```

Output:
Table created.

to see table structure:

```
DESC student
```

Output:

```
NAME                TYPE
----------------------------------------
SID                 NUMBER(4)
SNAME               VARCHAR2(10)
```

Adding a column [add m1]:

```
ALTER TABLE student ADD m1 NUMBER(3);
```

Output:
Table Altered.

```
DESC student
```

Output:

```
NAME                TYPE
----------------------------------------
SID                 NUMBER(4)
SNAME               VARCHAR2(10)
M1                  NUMBER(3)
```

Adding multiple columns [add m2, m3]:

```
ALTER TABLE student ADD(m2 NUMBER(3), m3 NUMBER(3));
```

Output:
Table Altered.

```
DESC student
```

Output:

```
NAME                TYPE
-----------------------------------------
SID                 NUMBER(4)
SNAME               VARCHAR2(10)
M1                  NUMBER(3)
M2                  NUMBER(3)
M3                  NUMBER(3)
```

Renaming column [m3 rename to maths]:

ALTER TABLE student RENAME COLUMN m3 TO maths;
Output:
Table Altered.

DESC student
Output:

```
NAME                TYPE
-----------------------------------------
SID                 NUMBER(4)
SNAME               VARCHAR2(10)
M1                  NUMBER(3)
M2                  NUMBER(3)
MATHS               NUMBER(3)
```

Dropping a column [drop maths column]:

ALTER TABLE student DROP COLUMN maths;
(or)
ALTER TABLE student DROP(maths);
Output:
Table Altered.

NOTE:
DROP COLUMN can drop one column only
DROP can drop one or multiple columns. For DROP parenthesis are mandatory.

DESC student
Output:

```
NAME                TYPE
-----------------------------------------
```

```
SID                     NUMBER(4)
SNAME                   VARCHAR2(10)
M1                      NUMBER(3)
M2                      NUMBER(3)
```

Dropping multiple columns [drop m1 and m2]:

```
ALTER TABLE student DROP(m1,m2);
Output:
Table Altered.

DESC student
Output:
NAME                    TYPE
----------------------------------------
SID                     NUMBER(4)
SNAME                   VARCHAR2(10)
```

Modifying Field size [sname field size modify to 20]:

```
ALTER TABLE student MODIFY sname VARCHAR2(20);
Output:
Table Altered.

DESC student
Output:
NAME                    TYPE
----------------------------------------
SID                     NUMBER(4)
SNAME                   VARCHAR2(20)
```

Can we decrease the field size?
YES. But, we can decrease up to max string length in column

```
SNAME       VARCHAR2(20)
---------------
KIRAN        5
NARESH       6              we can decrease up to 6
SAI          3
AMAR         4
```

Modifying data type [sid column data type modify to char(7)]:

ALTER TABLE student MODIFY sid CHAR(7);
Output:
Table Altered.

DESC student
Output:

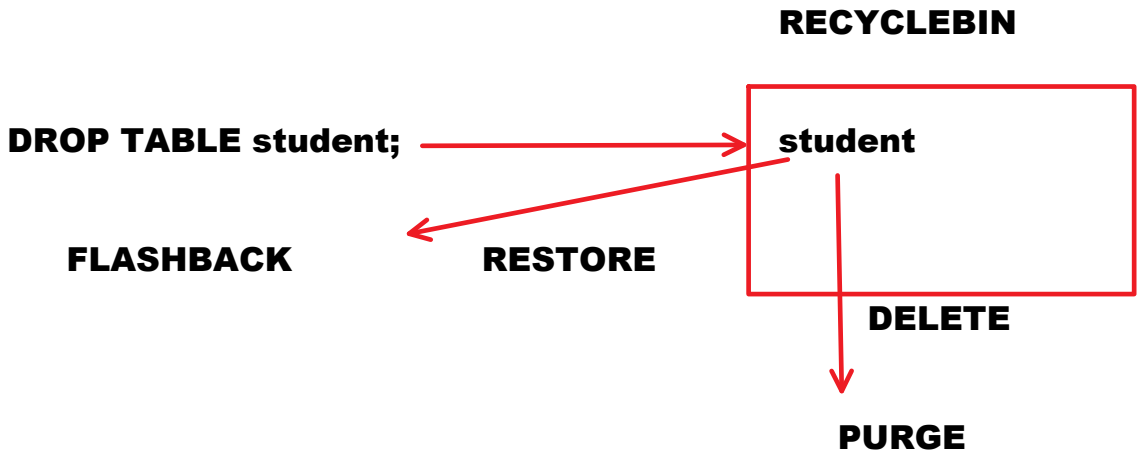
```
NAME                TYPE
-----------------------------------------
SID                 CHAR(7)
SNAME               VARCHAR2(20)
```

Note:
To modify the data type column must be empty.

DROP, FLASHBACK and PURGE

Friday, June 28, 2024 7:51 AM

DROP, FLASHBACK and PURGE:



RECYCLEBIN feature added in ORACLE 10g version.

BEFORE ORACLE 10g,
DROP TABLE student; --will be dropped permanently

DROP:

- **DROP command is used to delete the table.**
- **When table is dropped, it will be moved to RECYCLEBIN.**

Syntax:

```
DROP TABLE <table_name> [PURGE];
```

Example:

```
DROP TABLE student;
--it will be moved to recyclebin

DROP TABLE employee PURGE;
--table will be deleted permanently. it will not be moved to
```

recyclebin.

FLASHBACK:

- **FLASHBACK command introduced in ORACLE 10g version.**
- **It is used to restore the dropped table from RECYCLEBIN.**

Syntax:

```
FLASHBACK TABLE <table_name> TO BEFORE DROP
[RENAME TO <new_name>];
```

Example:

```
FLASHBACK TABLE student TO BEFORE DROP;
```

PURGE:

- **PURGE command introduced in ORACLE 10g version.**
- **It is used to delete the table from RECYCLEBIN.**
- **If table purged, it will be deleted permanently.**

Syntax:

```
PURGE TABLE <table_name>;
```

Example:

```
PURGE TABLE student;
```

to see RECYCLEBIN:

```
SHOW RECYCLEBIN
```

to delete a table permanent:

```
DROP TABLE student;
PURGE TABLE student;
```

(or)

```
DROP TABLE student PURGE;
```

emptying recyclebin:

```
PURGE RECYCLEBIN;
```

CASE-1:

```
CREATE TABLE t1(f1 INT);
insert into t1 values(1);
insert into t1 values(2);
COMMIT;

DROP TABLE t1;


CREATE TABLE t1(f1 INT,f2 CHAR);
insert into t1 values(1,'A');
insert into t1 values(2,'B');
commit;

DROP TABLE t1;

FLASHBACK TABLE t1
TO BEFORE DROP;
--it restores recently dropped t1  8:35
```

RECYCLEBIN

T1 8:35
T1 8:30

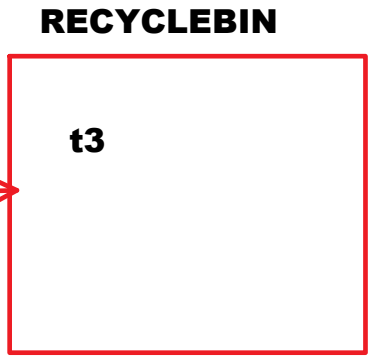
to restore older t1 [8:30]:

```
FLASHBACK TABLE "<recyclebin_name>"
TO BEFORE DROP;
```

Note:
Within SCHEMA, table name must be unique.

CASE-2:

```
CREATE TABLE t3(f1 INT);
insert into t3 values(1);
insert into t3 values(2);
commit;

DROP TABLE t3;

CREATE TABLE t3(f1 CHAR);
insert into t3 values('A');
commit;
```



DCL

Monday, July 1, 2024 7:35 AM

DCL:

- **DCL => Data Control Language.**
- **It deals with data accessibility.**
- **It is used to implement table level security.**

ORACLE SQL provides following DCL commands:

- **GRANT**
- **REVOKE**

GRANT:

- **It is used to grant permission on DB Objects to other users.**

  Syntax:

  ```
  GRANT <privileges_list>
  ON <DB_Object>
  TO <users_list>;
  ```

REVOKE:

- **REVOKE command is used to cancel the permissions on db objects from other users.**

  Syntax:

  ```
  REVOKE <privileges_list>
  ON <DB_Object>
  FROM <users_list>;
  ```

privileges_list
----
privilege => permission

SELECT
INSERT
UPDATE
DELETE
ALTER
.
.

Examples:

Granting read-only permission on emp table to c##userA:

```
GRANT select
ON emp
TO c##userA;
```

Granting DML permissions on emp table to c##userA:

```
GRANT insert, update, delete
ON emp
TO c##userA;
```

Granting all permissions on emp table to c##userA:

```
GRANT all
ON emp
TO c##userA;
```

Grant read-only permission on emp table to c##userA, c##userB and c##userC:

```
GRANT select
ON emp
TO c##userA, c##userB, c##userC;
```

Grant read-only permission on emp table to all users:

```
GRANT select
ON emp
TO public;
```

Cancel DML permissions on emp table from c##userA:

```
REVOKE insert, update, delete
ON emp
FROM c##userA;
```

Cancel SELECT permission on emp table from c##userA:

```
REVOKE select
ON emp
```

```
FROM c##userA;
```

Cancel all permissions on emp table from c##userA:

```
REVOKE select
ON emp
FROM c##userA;
```

Cancel all permissions on emp table from c##userA, c##userB, c##userC:

```
REVOKE all
ON emp
FROM c##userA, c##userB, c##userC;
```

Cancel all permissions on emp table from all users:

```
REVOKE all
ON emp
FROM public;
```

Example on GRANT and REVOKE:

Create 2 users c##userA, c##userB:

Login as DBA:

username: system
password: naresh

```
CREATE USER c##userA
IDENTIFED BY usera;

GRANT connect, resource, unlimited tablespace
TO c##userA;

CREATE USER c##userB
IDENTIFED BY userb;
```

```
GRANT connect, resource, unlimited tablespace
TO c##userB;
```

OPEN 2 SQL PLUS windows. Arrange them side by side

c##userA [GRANTOR]	c##userB [GRANTEE]

T1

F1	F2
1	A
2	B

```
CREATE TABLE t1
(
f1 NUMBER(4),
f2 VARCHAr2(10)
);

INSERT INTO t1 VALUES(1,'A');
INSERT INTO t1 VALUES(2,'B');
COMMIT;
```

```
SELECT * FROM c##userA.t1;
```

Output:

ERROR: table does not exist

```
GRANT select
ON t1
TO c##userB;
```

```
SELECT * FROM c##userA.t1;
```

Output:

F1	F2
1	A
2	B

GRANT insert, update, delete
ON t1
TO c##userB;

SELECT * FROM t1;
Output:

F1	F2
1	A
2	B

SELECT * FROM t1;
Output:
Output:

F1	F2
1	A
2	B
3	C

SELECT * FROM t1;

2	B

INSERT INTO c##userA.t1
VALUES(3,'C');
Output:
1 row created.

SELECT * FROM c##userA.t1;
Output:

F1	F2
1	A
2	B
3	C

COMMIT;

UPDATE c##userA.t1
SET f2='SAI'
WHERE f1=1;
Output:
1 row updated.

COMMIT;

SELECT * FROM t1;
Output:
Output:

F1	F2
1	SAI
2	B
3	C

DELETE FROM c##userA.t1
WHERE f1=3;
Output:
1 row deleted

COMMIT;

SELECT * FROM t1;
Output:
Output:

F1	F2
1	SAI
2	B

GRANT all
ON t1
TO c##userB;

ALTER TABLE c##userA.t1
ADD f3 DATE;
Output:
Table Altered

DESC t1;
Output:
F1
F2
F3

```
REVOKE insert,update,delete
ON t1
FROM c##userB;
```

INSERT => ERROR: insufficent privileges
DELETE => ERROR
UPDATE => ERROR

```
SELECT * FROM c##userA.t1;
```

Output:

F1	F2
1	A
2	B

```
REVOKE all
ON t1
FROM c##userB;
```

USER_TAB_PRIVS_MADE
USER_TAB_PRIVS_RECD

USER_TAB_PRIVS_MADE:

- it is a system table / built-in table / readymade table.
- it maintains all the permissions which are given by GRANTOR.

  to see list of privieges made by grantor:

```
SELECT grantee, table_name, privilege
FROM user_tab_privs_made;
```

USER_TAB_PRIVS_RECD:

- **it is a system table / built-in table / readymade table.**
- **it maintains all the permissions which are recieved by GRANTEE.**

to see list of permissions received by GRANTEE:

```
SELECT owner, table_name, grantor, privilege
FROM user_tab_privs_recd;
```

ALL => 12 permissions

SELECT
INSERT
UPDATE
DELETE
ALTER
FLASHBACK
DEBUG
QUERY REWRITE
ON COMMIT REFRESH
READ
REFERENCES
INDEX

Copying table and Copying Records

Tuesday, July 2, 2024 7:44 AM

Copying table:

Syntax:

```
CREATE TABLE <name>
AS
<SELECT QUERY>;
```

- Copying Table means, creating a new table from existing table.
- With SELECT QUERY result a new table will be created.

Examples:

Create exact copy of emp table with the name emp1:

EMP
8 columns
15 rows

EMP1
8 columns
15 rows

```
CREATE TABLE emp1
AS
SELECT * FROM emp;
```

Example-2:

EMP
8 columns
15 rows

EMP2
4 columns => empno, ename,job, sal
managers records

create a new table with the name emp2
from existing table emp
with 4 columns empno, ename, job, sal
and with managers records:

```
CREATE TABLE emp2
AS
SELECT empno, ename, job, sal
FROM emp
WHERE job='MANAGER';
```

Copying table Structure:

Syntax:

```
CREATE TABLE <name>
AS
SELECT <columns_list>
FROM <table_name>
WHERE <false_condition>;
```

False Condition

1=2 FALSE
'A'='B' FALSE
400=500 FALSE

Examples:

Create a new table with the name emp3
with emp table structure, without rows:

Emp
8 columns
15 rows

Emp3
8 columns
no rows

```
CREATE TABLE emp3
AS
SELECT * FROM emp
WHERE 1=2;
```

Example-2:

EMP
8 columns
15 rows

EMP4
EMPNO ENAME SAL
no rows

Create a new table with the name emp4 from existing table emp with 3 columns empno, ename, sal and without rows:

```
CREATE TABLE emp4
AS
SELECT empno, ename, sal
FROM emp
WHERE 1=2;
```

Copying Records:

Syntax:

Syntax:

```
INSERT INTO <table_name>
<SELECT QUERY>;
```

Examples:

EMP
8 columns
15 rows

copy →

EMP5
4 columns => empno, ename, job, sal
no rows

```
CREATE TABLE emp5
AS
SELECT empno, ename, job, sal
FROM emp
WHERE 1=2;
```

Copy all rows from emp table to emp5:

```
INSERT INTO emp5
SELECT empno, ename, job, sal FROM emp;
```

```
DELETE FROM emp5;
COMMIT;
```

Copy all managers records from emp to emp5:

EMP
8 columns
15 rows
managers

copy →

EMP5
EMPNO ENAME JOB SAL
no rows

8 columns
15 rows
managers

copy

EMPNO ENAME JOB SAL
no rows

```
INSERT INTO emp5
SELECT empno, ename, job, sal
FROM emp
WHERE job='MANAGER';
```

INSERT ALL

Tuesday, July 2, 2024 8:26 AM

INSERT ALL:

- **Introduced in ORACLE 9i.**
- **INSERT ALL command is used to copy one table data to multiple tables.**
- **It avoids of writing multiple INSERT commands.**
- **It can be used to perform ETL operations.**
- **E=> Extract, T => transfer, L => Load**
- **INSERT ALL can be used in 2 ways. They are:**
  - **Unconditional INSERT ALL**
  - **Conditional INSERT ALL**

Unconditional INSERT ALL:

Syntax:

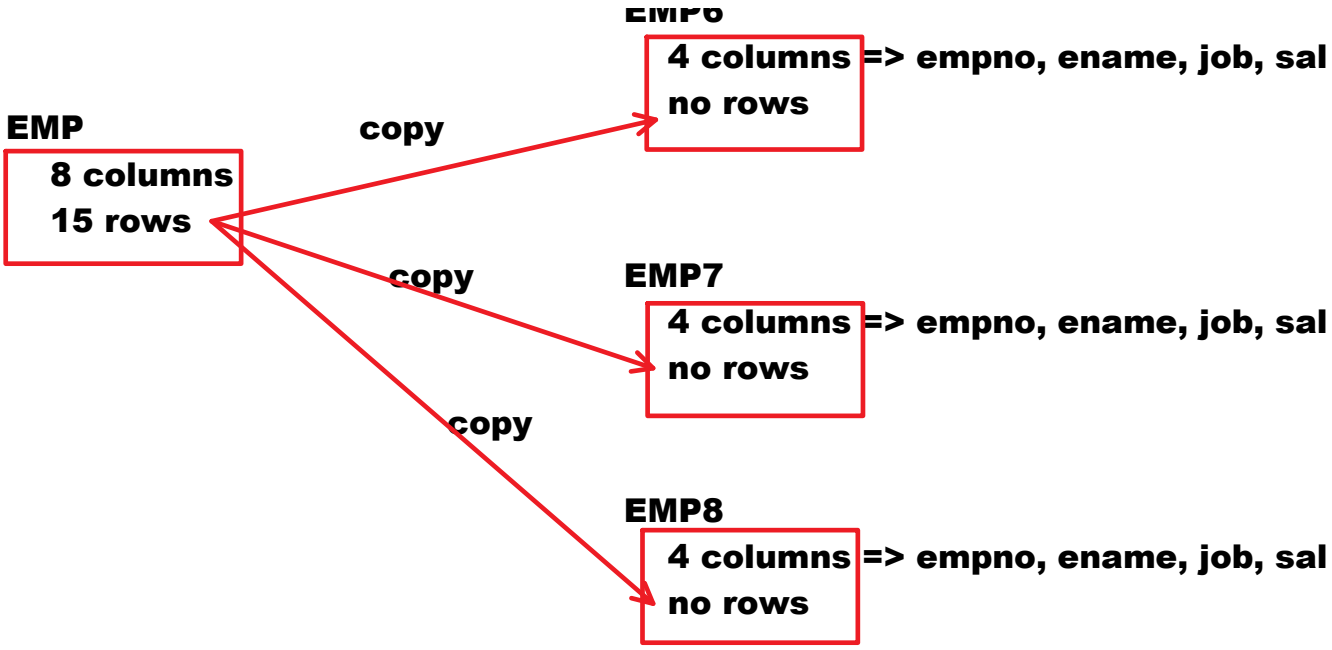
```
INSERT ALL
INTO <table_name>(<columns_list>) VALUES(<values_list>)
INTO <table_name>(<columns_list>) VALUES(<values_list>)
.
.
<SELECT QUERY>;
```

Example on Unconditional INSERT ALL:

EMP6

4 columns => empno, ename, job, sal

no rows



Copy emp table all rows to emp6, emp7, emp8:

create the tables emp6, emp7, emp8 with 4 columns without rows:

```
CREATE TABLE emp6
AS
SELECT empno, ename, job, sal
FROM emp
WHERE 1=2;

CREATE TABLE emp7
AS
SELECT empno, ename, job, sal
FROM emp
WHERE 1=2;

CREATE TABLE emp8
AS
SELECT empno, ename, job, sal
FROM emp
WHERE 1=2;
```

Copy emp table all rows to emp6, emp7, emp8:

```
INSERT ALL
INTO emp6 VALUES(empno, ename, job, sal)
INTO emp7 VALUES(empno, ename, job, sal)
INTO emp8 VALUES(empno, ename, job, sal)
SELECT empno,ename,job,sal FROM emp;
```

Output:

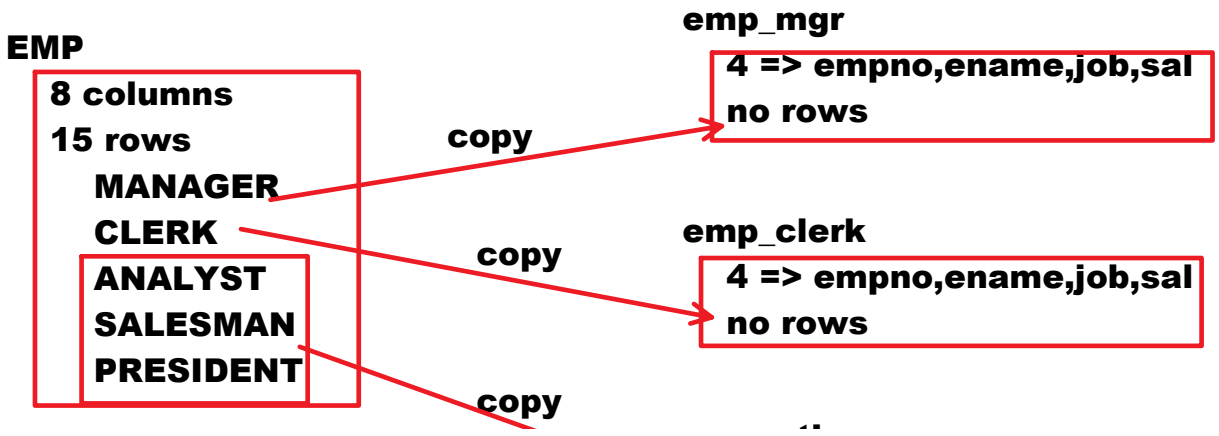
45 rows created.

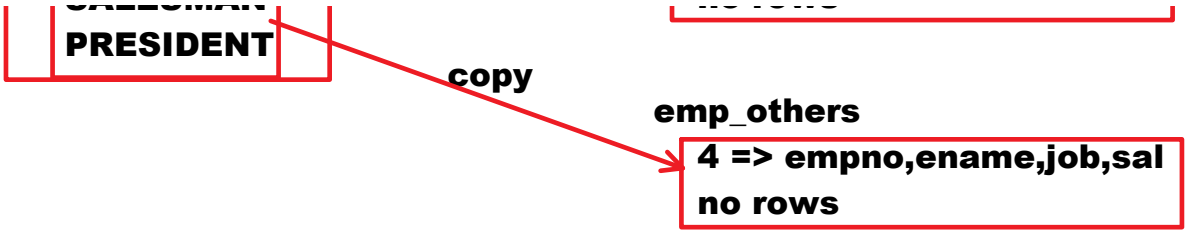
Conditional INSERT ALL:

Syntax:

```
INSERT ALL
WHEN <condition1> THEN
INTO <table_name>(<columns_list>) VALUES(<values_list>)
WHEN <condition2> THEN
INTO <table_name>(<columns_list>) VALUES(<values_list>)
.
.
ELSE
INTO <table_name>(<columns_list>) VALUES(<values_list>)
<SELECT QUERY>;
```

Example on Conditional INSERT ALL:





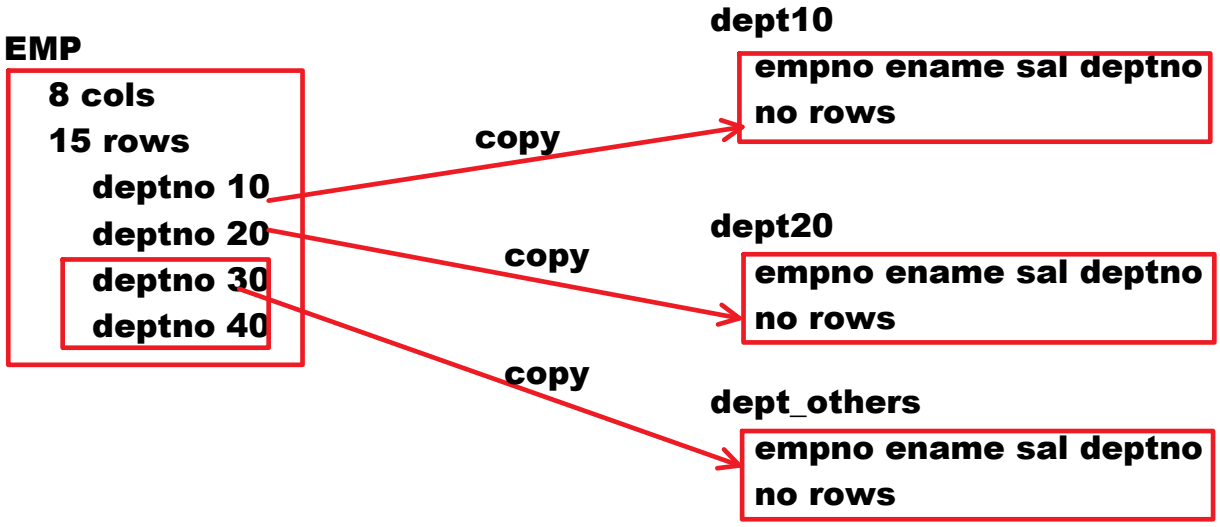
create emp_mgr, emp_clerk and emp_others:

```
CREATE TABLE emp_mgr
AS
SELECT empno,ename,job,sal
FROM emp
WHERE 1=2;

CREATE TABLE emp_clerk
AS
SELECT empno,ename,job,sal
FROM emp
WHERE 1=2;

CREATE TABLE emp_others
AS
SELECT empno,ename,job,sal
FROM emp
WHERE 1=2;
```

copy managers records to emp_mgr,
clerks records to emp_clerk,
other than managers and clerks copy to emp_others:

```
INSERT ALL
WHEN job='MANAGER' THEN
INTO emp_mgr VALUES(empno,ename,job,sal)
WHEN job='CLERK' THEN
INTO emp_clerk VALUES(empno,ename,job,sal)
ELSE
INTO emp_others VALUES(empno,ename,job,sal)
SELECT * FROM emp;
```

Assignment:



WHEN hiredate BETWEEN '1-JAN-1980'
AND '31-DEC-1980' THEN

emp1980

emp1981

emp_others

MERGE

Wednesday, July 3, 2024 8:02 AM

MERGE: replica => duplicate copy

Branch Office s.cid = t.cid **Head Office**

CUSTOMER1 s

CID	CNAME	CCITY
1001	~~A~~ ABC	~~HYD~~ BLR
1002	B	BLR
1003	C	MUM
1004	D	PUN
1005	E	CHN

CUSTOMER2 t => replica

CID	CNAME	CCITY
1001	A	HYD
1002	B	BLR
1003	C	MUM

MERGE:

- Introduced in ORACLE 9i.
- MERGE = UPDATE + INSERT
- MERGE is a combination of UPDATE and INSERT commands.
- It is used to apply one table changes to its replica.

Syntax:

```
MERGE INTO <target table_name> <target_table_alias>
USING <source_table_name> <source_table_alias>
ON(<condition>)
WHEN matched THEN
UPDATE query
WHEN not matched THEN
INSERT query;
```

Example on MERGE:

s.cid = t.cid

CUSTOMER1 s

CID	CNAME	CCITY
1001	~~A~~ ABC	~~HYD~~ BLR
1002	B	BLR
1003	C	MUM
1004	D	PUN
1005	E	CHN

CUSTOMER2 t => replica

CID	CNAME	CCITY
1001	A	HYD
1002	B	BLR
1003	C	MUM

```
CREATE TABLE customer1
(
cid NUMBER(4),
cname VARCHAR2(10),
ccity CHAR(3)
);

INSERT INTO customer1 VALUES(1001,'A','HYD');
INSERT INTO customer1 VALUES(1002,'B','BLR');
INSERT INTO customer1 VALUES(1003,'C','MUM');

COMMIT;

CREATE TABLE customer2
AS
SELECT * FROM customer1;
```

1004	D	PUN
1005	E	CHN

```
INSERT INTO customer1 VALUES(1004,'D','PUN');
INSERT INTO customer1 VALUES(1005,'E','CHN');
```

COMMIT;

CID	CNAME	CCITY
1001	~~A~~ ABC	~~HYD~~ BLR

UPDATE customer1
SET cname='ABC', ccity='BLR'
WHERE cid=1001;

COMMIT;

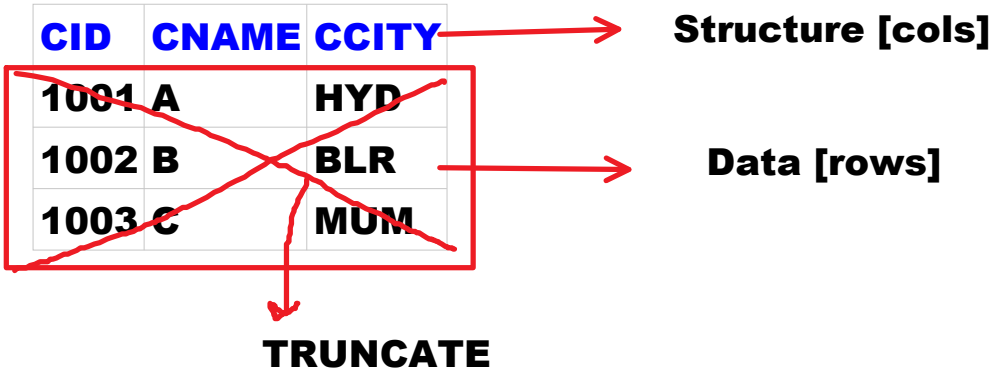
Apply cutomer1 table changes to its replica customer2:

MERGE INTO customer2 T
USING customer1 S
ON(S.cid=T.cid)
WHEN matched THEN
UPDATE SET t.cname=s.cname, t.ccity=s.ccity
WHEN not matched THEN
INSERT VALUES(s.cid, s.cname, s.ccity);
Output:
5 rows merged.

TRUNCATE

Wednesday, July 3, 2024 8:54 AM

TRUNCATE:



- **TRUNCATE command is used to delete all rows from table with good performance.**
- **It permanently deletes the rows. we cannot recollect them.**

Syntax:

```
TRUNCATE TABLE <table_name>;
```

Example:

```
TRUNCATE TABLE customer1;
--all rows will be deleted permanently

DESC customer1
Output:
NAME
------------
CID
CNAME
CCITY
```

Note:

TRUNCATE deletes entire table data
It does not delete table structure

Differences b/w TRUNCATE and DELETE:

TRUNCATE	DELETE
• It is DDL command	• it is DML command
• It is auto committed	• It is not auto committed
• We cannot delete single record or specific group of records. Just we can delete all records.	• We can delete single record or specific group of records or all records.
• It cannot be rolled back.	• it can be rolled back.
• WHERE clause cannot be used here.	• WHERE clause can be used here.
• it is faster.	• It is slower.
• It deletes block by block [page by page].	• It deletes row by row.

DATABASE
TABLESPACES
SEGMENTS
EXTENTS
BLOCKS

DATA

RENAME:

It is used to rename the table.

Syntax:

```
RENAME <old_name> TO <new_name>;
```

Example:

```
RENAME emp TO e;
```

Output:

Table renamed.

By default,
All DDL commands are auto committed.
All DML commands are not auto committed.

DDL = DDL + COMMIT
CREATE = CREATE + COMMIT

```
CREATE TABLE t10(f1 INT);   => DB  => committed
INSERT INTO t10 VALUES(1);
INSERT INTO t10 VALUES(2);
CREATE TABLE t11(f1 CHAR);   => committed
INSERT INTO t10 VALUES(3);
INSERT INTO t10 VALUES(4);
ROLLBACK;
```

DUAL

Friday, July 5, 2024 7:43 AM

DUAL:

- **DUAL is a readymade table / built-in table.**
- **DUAL table has 1 column and 1 row.**

SELECT * FROM dual;

Output:

DUAL

DUMMY
X

DESC dual

Output:

```
NAME          TYPE
----------    -------------
DUMMY         VARCHAR2(1)
```

- **When we want to work with non-table data or when we want to get 1 value as the result then we use DUAL.**

SELECT 100+200 FROM dual;

Output:

300 => dual table has 1 row

SELECT 100+200 FROM emp;

Output:
300
300
300
.
.
300

15 300s => emp table has 15 rows

In ORACLE 23 AI, they made FROM clause as optional.

Till ORACLE 21C,
SELECT 100+200; ERROR: FROM not found

In ORACLE 23AI,
SELECT 100+200; --300

SELECT lower('RAJU'); --raju

Built-In Functions

Thursday, July 4, 2024 8:16 AM

Built-In Functions:

- **To make our actions easier ORACLE software developers defined some functions and placed them in ORACLE DB. These functions are called "Built-In Functions / Predefined Functions / Readymade Functions".**

- **ORACLE SQL provides built-in functions. Those can be categorized as following:**
  - **String Functions**
  - **Conversion Functions**
  - **Aggregate Functions / Group Functions / Multi Row Functions**
  - **Number Functions**
  - **Date Functions**
  - **Analytic Functions / Window Functions**
  - **Special Functions**

String Functions:

lower()	Lpad()	Substr()	Chr()
upper()	Rpad()	Instr()	ASCII()
initcap()			Soundex()
	Ltrim()	Replace()	
length()	Rtrim()	Translate()	
concat()	Trim()	Reverse()	

LOWER():

- **It is used to convert string to lower case.**

Syntax:

Lower(<string>)

Examples:

Lower('RAJU')	raju
Lower('RAJ KUMAR')	raj kumar

SELECT lower('RAJU') FROM dual;

Output:

raju

upper():

- **it is used to convert string to upper case.**

Syntax:

upper(<string>)

Examples:

upper('raju')	RAJU
upper('ravi teja')	RAVI TEJA

Initcap() [initial capital]:

- **It is used to get every word's starting letter as capital.**

Syntax:

initcap(<string>)

Example:

initcap('RAJ KUMAR VARMA')	Raj Kumar Varma

Examples:

Display all emp names and salaries. display emp names in lower case:

```
SELECT lower(ename), sal FROM emp;

LOWER(ENAME)          SAL
-----------------------------------------------------
smith                  800
allen                 1600


SELECT lower(ename) AS ename, sal FROM emp;

ENAME                 SAL
-----------------------------------------------------
smith                  800
allen                 1600
```

Display BLAKE record when we don't know exact case:

```
SELECT * FROM emp
WHERE lower(ename)='blake';
```

```
ENAME          lower(ename)='blake'
-------------  ------------------------------------
SMITH          lower('SMITH') => smith =  blake  F
ALLEN          lower('ALLEN') => allen  = blake   F
BLAKE          lower('BLAKE') => blake = blake   T
```

Display the emp names and salaries.
Display all emp names in lower case:

```
SELECT lower(ename) AS ename, sal FROM emp;
```

Modify all emp names to lower case:

```
UPDATE emp
SET ename=lower(ename);
```

length():

- **it is used to find length of the string.**
- **string length => no of chars in string.**

Syntax:

```
length(<string>)
```

Examples:

length('RAJU')	4
length('RAVI TEJA')	9

Display the emp records whose names are having 4 letters:

```
SELECT ename, sal
```

```
FROM emp
WHERE length(ename)=4;
```

(or)

```
SELECT ename, sal
FROM emp
WHERE ename LIKE '____';
```

Display the emp records whose names are having 14 letters:

```
SELECT ename, sal
FROM emp
WHERE length(ename)=14;
```

Concat():

- **It is used to combine 2 strings.**

Syntax:

```
concat(<string1>, <string2>)
```

Examples:

concat('RAJ', 'KUMAR')	RAJKUMAR
concat('RAJ', 'KUMAR', 'VARMA')	ERROR
Concat(concat('RAJ', 'KUMAR'), 'VARMA')	RAJKUMARVARMA
'RAJ' || ' ' || 'KUMAR' || ' ' || 'VARMA'	RAJ KUMAR VARMA

Example:

EMPLOYEE

EMPID	FNAME	LNAME
1234	SAI	KRISHNA
1235	RAVI	TEJA

```
ENAME
------------
Sai Krishna
Ravi Teja
```

```
ALTER TABLE employee
ADD ename VARCHAR2(20);

UPDATE employee
SET ename= Initcap(fname || ' ' || lname);
```

Substr():

- **It is used to get sub string from the string.**
- **Sub String => part of the string.**

Syntax:

Substr(<string>, <position> [, <no_of_chars>])

Examples:

1	2	3	4	5	6	7	8	9
R	A	J		K	U	M	A	R

Substr('RAJ KUMAR', 5)	KUMAR
Substr('RAJ KUMAR', 1, 3)	RAJ
Substr('RAJ KUMAR',6,3)	UMA
Substr('RAJ KUMAR',6)	UMAR

2nd argument [position] can be -ve.

+VE	From Left side
-VE	From Right side

1	2	3	4	5	6	7	8	9
R	A	J		K	U	M	A	R
-9	-8	-7	-6	-5	-4	-3	-2	-1

Substr('RAJ KUMAR', -4)	UMAR
Substr('RAJ KUMAR', -4, 3)	UMA
Substr('RAJ KUMAR', -9, 3)	RAJ

Credit Card Bill => mail

your password:
your name's first 4 chars
your mobile number's last 4 digits

cname	mobile
VIJAY KAUMAR	9123456789

Substr(cname, 1,4) || Substr(mobile, -4, 4)

VIJA6789

Generate mail ids to all emps by taking emp name's first 3 chars, empno's last 3 digits as user name for the domain tcs.com:

EMP

EMPNO	ENAME
7369	SMITH
7499	ALLEN

MAIL_ID

SMI369@tcs.com
ALL499@tcs.com

```
ALTER TABLE emp ADD mail_id VARCHAR2(30);

UPDATE emp
SET mail_id = Substr(ename,1,3) || Substr(empno,-3,3) || '@tcs.com';
```

Display the emp records whose names are started with S:

```
SELECT *
FROM emp
WHERE Substr(ename,1,1)='S';

(or)

SELECT *
FROM emp
WHERE ename LIKE 'S%';
```

Display the emp names whose names are started vowel:

```
SELECT *
FROM emp
WHERE substr(ename,1,1) IN('A','E','I','O','U');
```

Display the emp records whose names are started and ended with same letter:

```
SELECT * FROM emp
WHERE substr(ename,1,1) = substr(ename,-1,1);
```

Lpad() & Rpad():

Lpad():
it is used to fill specified char set at left side.

Syntax:

Lpad(<string>, <size> [, <char_set>])

char_set = char / chars

3rd argument	default value => space

Rpad():
it is used to fill specified char set at right side.

Syntax:

Rpad(<string>, <size> [, <char_set>])

Examples:

Lpad('RAJU', 10, '*')	******RAJU
Rpad('RAJU', 10, '*')	RAJU******
Lpad('SAI',8,'@')	@@@@@SAI
Rpad('SAI',8,'@')	SAI@@@@@
Lpad('RAJU',10)	6spacesRAJU
Rpad('RAJU',10)	RAJU6spaces

10-4 = 6

8-3 = 5

Lpad('SAI',8,'@#')	@#@#@SAI
Lpad('A',6,'A')	AAAAAA

8-3 = 5

Lpad('X',8,'X')	XXXXXXXX

Example:
Display output as following if acno is 1234567891:
amount debited from acno XXXXXX7891

```
SELECT 'amount debited from acno ' || Lpad('X',6,'X') ||
Substr('1234567891',-4) FROM dual;
```

Ltrim(), Rtrim() and Trim():

Ltrim():

- It is used to remove unwanted chars from left side

Syntax:

Ltrim(<string> [, <char_set>])

2nd arg	default value => space

Rtrim():

- It is used to remove unwanted chars from right side

Syntax:

Rtrim(<string> [, <char_set>])

2nd arg	default value => space

Ltrim('*****RAJU*****', '*')	RAJU*****
Rtrim('*****RAJU*****', '*')	*****RAJU
Ltrim(' RAJU ')	RAJU3spaces
Rtrim(' RAJU ')	3spacesRAJU

Trim():

It can be used to remove unwanted chars from left side or right side or both sides.

Syntax:

Trim(leading / trailing / both <char> from <string>)

Examples:

Trim(leading '*' FROM '*****RAJU*****')	RAJU*****
Trim(trailing '*' FROM '*****RAJU*****')	*****RAJU
Trim(both '*' FROM '*****RAJU*****')	RAJU
Trim(' RAJU ')	RAJU

default side => both
default char => space

Instr():

- **It is used to check whether the sub string is existed in string or not.**
- **If sub string existed in string, it returns position number.**
  If sub string is not existed I string, it returns 0.

Syntax:

Instr(<string>, <sub_string> [, <position>, <occurrence>])

1	2	3	4	5	6	7	8	9
R	A	V	I		T	E	J	A

3rd arg default position	1
4th arg default occurrence	1

Examples:

Instr('RAVI TEJA', 'TEJA')	6
Instr('RAVI TEJA', 'RAVI')	1
Instr('RAVI TEJA', 'I TE')	4
Instr('RAVI TEJA', 'SAI')	0

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
T	H	I	S		I	S		H	I	S		W	I	S	H

Instr('THIS IS HIS WISH', 'IS')	3
Instr('THIS IS HIS WISH', 'IS', 1, 2)	6
Instr('THIS IS HIS WISH', 'IS', 4, 3)	14
Instr('THIS IS HIS WISH', 'IS', 4, 4)	0

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
T	H	I	S		I	S		H	I	S		W	I	S	H
-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Instr('THIS IS HIS WISH', 'IS', -1)	14
Instr('THIS IS HIS WISH', 'IS', -1,3)	6
Instr('THIS IS HIS WISH', 'IS', -4, 3)	3

Display the emp records whose names are having AM chars:

```
SELECT ename, sal, Instr(ename, 'AM')
FROM emp
WHERE Instr(ename, 'AM')>0;

(or)

SELECT ename, sal
FROM emp
WHERE ename LIKE '%AM%';
```

Instr(ename, 'AM')

JAMES => 2
ADAMS => 3
AMAR => 1
SMITH => 0

Example:

EMP60

EMPID	ENAME
1234	RAJ KUMAR
1235	VIJAY KUMAR
1236	SAI KRISHNA

FNAME	LNAME
RAJ	KUMAR
VIJAY	KUMAR
SAI	KRISHNA

```
CREATE TABLE emp60
(
empid NUMBER(4),
ename VARCHAR2(20)
);
```

```
INSERT INTO emp60 VALUES(1234,'RAJ KUMAR');
INSERT INTO emp60 VALUES(1235,'VIJAY KUMAR');
INSERT INTO emp60 VALUES(1236,'SAI KRISHNA');
COMMIT;


ALTER TABLE emp60 ADD(fname VARHCAR2(10),
lname VARCHAR2(10));

UPDATE emp60
SET fname=Substr(ename, 1, Instr(ename, ' ')-1),
lname= Substr(ename, Instr(ename, ' ')+1);

COMMIT;

ALTER TABLE emp60 DROP(ename);
```

Display the emp records whose names are having _:

```
SELECT ename, sal
FROM emp
WHERE instr(ename,'_')>0;

(or)

SELECT ename, sal
FROM emp
WHERE ename LIKE '%\_%' ESCAPE '\';

(or)

SELECT ename, sal
FROM emp
WHERE ename LIKE '%$_%' ESCAPE '$';

(or)

SELECT ename, sal
FROM emp
WHERE ename LIKE '%#_%' ESCAPE '#';
```

Display the emp records whose names are having %:

```
SELECT ename, sal
FROM emp
WHERE Instr(ename, '%')>0;
```

(or)

```
SELECT ename, sal
FROM emp
WHERE ename LIKE '%\%%' ESCAPE '\';
```

Replace() & Translate():

Replace():

It is used to replace search string with replace string.

Syntax:

Repalce(<string>, <search string>, <replace string>)

Examples:

Replace('SAI KRISHNA', 'SAI', 'RAVI')	RAVI KRISHNA
Replace('SAI KRISHNA SAI TEJA', 'SAI', 'RAVI')	RAVI KRISHNA RAVI TEJA

Translate():

- it is used to replace the characters.

Syntax:

Translate(<string>, <search_char_set>, <replace_char_set>)

Examples:

Replace('SAI KRISHNA', 'SAI', 'XYZ')	XYZ KRISHNA
Translate('SAI KRISHNA' 'SAI', 'XYZ') S => X A => Y I => Z	XYZ KRZXHNY

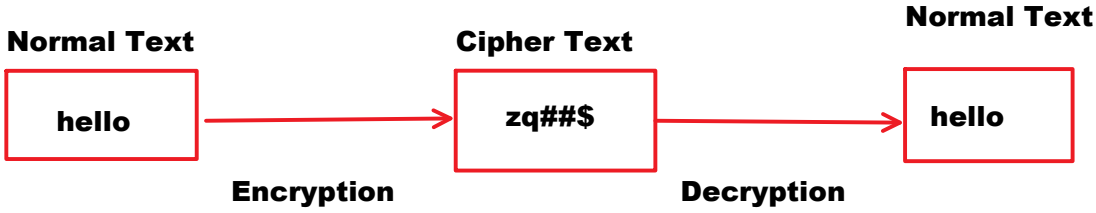
Replace('abcabcaabbccabc', 'abc', 'XYZ')	XYZXYZaabbccXYZ

Translate('abcabcaabbccabc', 'abc','XYZ') a => X b => Y c => Z	XYZXYZXXYYZZXYZ

Difference b/w Replace() & Translate():

Replace()	It is used to replace the strings
Translate()	It is used to replace the chars

Translate() can be used to encrypt and decrypt the data.



Display the emp names and salaries.
Encrypt salaries as following:

0	1	2	3	4	5	6	7	8	9
@	Z	Q	$	^	W	B	*	!	%

```
SELECT ename, translate(sal, '0123456789', '@ZQ$^WB*!%') AS sal
FROM emp;
```

Remove special chars from following string:
RA#@^JU

```
SELECT Replace('RA#@^JU', '#@^', '')
FROM dual;
```

Reverse():

- **it returns reverse string**

Syntax:

Reverse(<string>)

Example:

Reverse('SAI')	IAS

ASCII():

It returns ASCII value of specified char

Syntax:

ASCII(<char>)

Examples:

ASCII('A')	65
ASCII('Z')	90
ASCII('a')	97

Chr():

- **It returns character of specified ASCII value.**

Examples:

Chr(65)	A
Chr(97)	a

Soundex():

- **It is used to retrieve the data based on sounds.**
- **When we don't exact spelling we retrieve data based on sounds.**

Syntax:

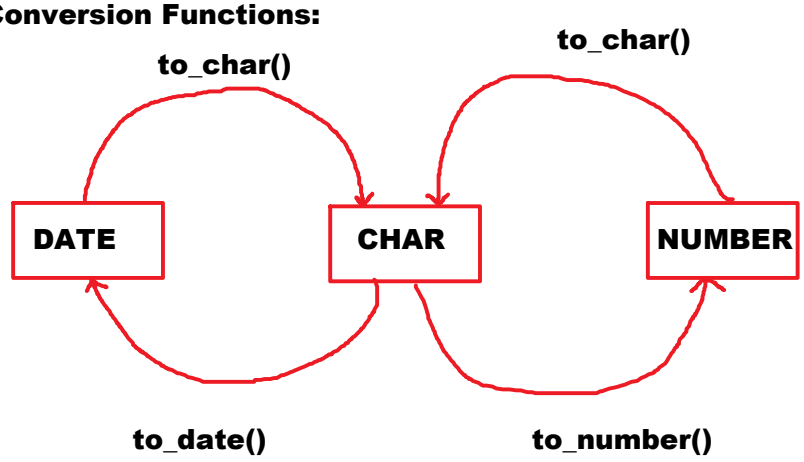
Soundex(<string1>) = Soundex(<string2>)

Examples:

Display BLAKE record:

```
SELECT * FROM emp
WHERE soundex(ename)=soundex('blek');
```

Conversion Functions:



To_Char() [Date to Char]:

- **It is used to convert date to char [string].**
- **It is used change date formats.**
- **It can be also used to extract part of the date.**

Syntax:

To_Char(<date>, <format>)

FORMAT	PURPOSE	EXAMPLE sysdate => 9-JUL-24	OUTPUT
YYYY	year 4 digits	to_char(sysdate, 'YYYY')	2024
YY	year last 2 digits	to_char(sysdate, 'YY')	24
YEAR / year	year in words	to_char(sysdate, 'YEAR') to_char(sysdate, 'year')	TWENTY TWENTY-FOUR twenty twenty-four
MM	month number	to_char(sysdate, 'MM')	07
MON	short month name	to_char(sysdate, 'MON')	JUL
MONTH	full month name	to_char(sysdate, 'MONTH')	JULY

D	day num in week	to_char(sysdate, 'D')	3
DD	day num in month	to_char(sysdate, 'DD')	09
DDD	day num in year	to_char(sysdate, 'DDD')	191 31+29+31+30+31+30+9 = 191
DY	short weekday name	to_char(sysdate,'DY')	TUE
DAY	full weekday name	to_char(sysdate, 'DAY')	TUESDAY
Q	quarter num 1 => jan-mar 2 => apr-jun 3 => jul-sep 4 => oct-dec	to_char(sysdate, 'Q')	3
CC	century num	to_char(sysdate, 'CC')	21
HH / HH12	hours part in 12 hrs format		
HH24	hours part in 24 hrs format		
MI	minutes part		
SS	seconds part		
FF	fractional seconds		
AM / PM	AM or PM		

Display current system time in 12hrs format:

```
SELECT to_char(sysdate, 'HH:MI:SS AM')
FROM dual;
```

Display current system time in 24hrs format:

```
SELECT to_char(sysdate, 'HH24:MI:SS')
FROM dual;
```

Examples:

Display the emp records who joined in 1982:

```
SELECT ename, hiredate
FROM emp
WHERE to_char(hiredate,'YYYY')=1982;
```

Display the emp records who joined in 1980, 1982, 1984:

```
SELECT ename, hiredate
FROM emp
WHERE to_char(hiredate,'YYYY') IN(1980,1982,1984);
```

Display the emp records who joined in DECEMBER month:

```
SELECT ename, hiredate
FROM emp
WHERE to_char(hiredate,'MM')=12;
(or)
SELECT ename, hiredate
FROM emp
WHERE to_char(hiredate,'MON')='DEC';
(or)
SELECT ename, hiredate
FROM emp
WHERE RTRIM(to_char(hiredate,'MONTH'))='DECEMBER';
```

Display the emp records who joined in JAN, MAY and DEC:

```
SELECT ename, hiredate
FROM emp
WHERE to_char(hiredate,'MM') IN(1,5,12);
```

Display the emp records who joined in 4th qrtr:

```
SELECT ename, hiredate
FROM emp
WHERE to_char(hiredate,'Q')=4;
```

Display the emp records who joined in 1st and 4th qrtr:

```
SELECT ename, hiredate
FROM emp
WHERE to_char(hiredate,'Q') IN(1,4);
```

Display the emp records who joined on SUNDAY:

```
SELECT ename, hiredate
FROM emp
WHERE to_char(hiredate,'D')=1;
```

(or)

```
SELECT ename, hiredate
FROM emp
WHERE to_char(hiredate, 'DY') = 'SUN';
```

```
SELECT ename, hiredate
FROM emp
WHERE to_char(hiredate,'DAY')='SUNDAY';
```

SUNDAY3spaces = SUNDAY FALSE

Output:

no rows selected

to_char(hiredate,'DAY')

```
SUNDAY3spaces   9
TUESDAY2spaces 9
WEDNESDAY        9        max str length
.
.
SATURDAY1space  9
```

```
SELECT ename, hiredate
FROM emp
WHERE RTRIM(to_char(hiredate, 'DAY')) = 'SUNDAY';
```

RTRIM(SUNDAY3spaces)
SUNDAY = SUNDAY TRUE

Display the emp names and hiredates.
Display hiredates IND date format:

```
SELECT ename, to_char(hiredate, 'DD/MM/YYYY') AS hiredate
FROM emp;
```

To_Char() [number to char]:

- **It can be used to convert number to char [string].**
- **To apply currency symbols, currency names, thousand separator, decimal point and decimal places we need to convert number to char.**

Syntax:

To_Char(<number> [, <format> , <NLS_parameters>])

Examples:

To_Char(123)	'123'
To_Char(123.45)	'123.45'

FORMAT	PURPOSE
L	currency symbol [$]
C	currency name [USD]
. / D	Decimal Point
, / G	Thousand separator
9	Digit

To_Char(5000, 'L9999.99')	$5000.00
TO_Char(5000, 'C9,999.99')	USD5,000.00

Display all emp names and salaries.
Apply currency symbol $, thousand separator, decimal point and decimal places

```
SELECT ename, To_Char(sal, 'L99,999.99') AS sal
FROM emp;
```

NLS PARAMETERS	DEFAULT VALUE
NLS_CURRENCY	$
NLS_ISO_CURRENCY	AMERICA

NLS => national Language Support

Display all emp names and salaries.
Apply currency symbol ¥, thousand separator, decimal point and decimal places:

```
SELECT ename, TO_Char(sal, 'L99999.99', 'nls_currency=¥') AS sal
FROM emp;
```

Display all emp names and salaries.
Apply currency name JPY, thousand separator, decimal point and decimal places:

```
SELECT ename, TO_Char(sal, 'C99999.99', 'nls_iso_currency=JAPAN') AS sal
FROM emp;
```

To_Date():

- It is used to convert string to date.
- To insert date values for explicit convertion we use it.

Syntax:

To_Date(<string> [, <format>])

Examples:

To_Date('25-DEC-2023')	25-DEC-23
To_Date('25 DECEMBER 2023')	25-DEC-23
To_Date('DECEMBER 25 2023')	ERROR
To_Date('DECEMBER 25 2023', 'MONTH DD YYYY')	25-DEC-23
To_date('25/12/2023')	ERROR

To_Date('25/12/2023', 'DD/MM/YYYY')	25-DEC-23

Example:

```
CREATE TABLE t10(f1 DATE);

INSERT INTO t10 VALUES('25-DEC-2023');
                                 string

F1                                implicit conversion
-------
25-DEC-23    date

Note:
implicit conversion degrades the performance

INSERT INTO t10 VALUES(to_date('17-AUG-2023'));
                                        string
F1
---------
17-AUG-23   date


INSERT INTO t10 VALUES(to_date('27/10/2023', 'DD/MM/YYYY'));
                                      string

 F1
 -------
 27-OCT-23   date


 INSERT INTO t10 VALUES(to_date('&d/&m/&y', 'DD/MM/YYYY'));
 Output:
 Enter value for d: 25
 Enter value for m:11
 Enter value for y:2020


 /
 Enter value for d: 17
 Enter value for m:2
 Enter value for y:2021
```

Example:

Extract year part from today's date:

```
SELECT to_char(sysdate, 'YYYY') FROM dual;
Output:
2024
```

date

Extract year part from 25-DEC-2023:

```
SELECT to_char('25-DEC-2023', 'YYYY') FROM dual;
Output:
ERROR
```

string

```
SELECT to_char(To_Date('25-DEC-2023'), 'YYYY') FROM
dual;
Output:
2023
```

find today's weekday:

```
SELECT to_char(sysdate, 'DAY') FROM dual;
```

find the weekday on which INDIA got independence:

```
SELECT to_char('15-AUG-1947', 'DAY') FROM dual;
Output:
ERROR

SELECT to_char(to_Date('15-AUG-1947'), 'DAY')
FROM dual;
Output:
FRIDAY
```

To_Number():

- **It is used to convert string to number.**
- **String must be numeric string only.**

Syntax:

To_Number('123')	123

To_Number('123.45')	123.45
To_Number('$5000.00')	ERROR
To_Number('$5000.00', 'L9999.99')	5000

Aggregate Functions / Group Functions / Multi Row Functions:

```
F1
-----
10
20
30
```

sum(f1)	10+20+30 = 60
avg(f1)	60/3 = 20
max(f1)	30
min(f1)	10
count(*)	3

ORACLE SQL provides following Aggregate Functions:

- sum()
- avg()
- max()
- min()
- count()

sum():

- it us used to find sum of a set of values.

  Syntax:

  sum(<column>)

  Examples:

  Find sum of salaries of all emps:

  ```
  SELECT sum(sal) FROM emp;
  ```

  Find sum of salaries of managers:

  ```
  SELECT sum(sal) FROM emp
  WHERE job='MANAGER';
  ```

  Find sum of salaries of emps who are working in

deptno 30:

```
SELECT sum(Sal) FROM emp
WHERE deptno=30;
```

avg():

- **It is used to find average of a set of values.**

Syntax:

```
avg(<column>)
```

Examples:

Find avrg sal of all emps:

```
SELECt avg(sal) FROM emp;
```

Find avrg sal of all managers:

```
SELECT avg(Sal) FROM emp
WHERE job='MANAGER';
```

max():

It is used to find max value in a set of values.

Syntax:

```
max(<column>)
```

Examples:

Find max salary in all emps:

```
SELECT max(sal) FROM emp;
```

Find max salary in all CLERKS:

```
SELECT max(Sal) FROM emp
WHERE job='CLERK';
```

min():

It is used to find min value in a set of values.

Syntax:

```
min(<column>)
```

Examples:

Find min salary in all emps:

```
SELECT min(sal) FROM emp;
```

Find min salary in all CLERKS:

```
SELECT min(Sal) FROM emp
WHERE job='CLERK';
```

count():

- **it is used to find number of records or number of column values.**

Syntax:

```
count(* / <column>)
```

Examples:

Find number of records in emp table:

```
SELECT count(*) FROM emp;
```

Find how many emps are getting commission:

```
SELECT count(comm) FROM emp;
```

Find number of clerks in emp table:

```
SELECT count(*) FROM emp
WHERE job='CLERK';
```

Find number of emps in deptno 30:

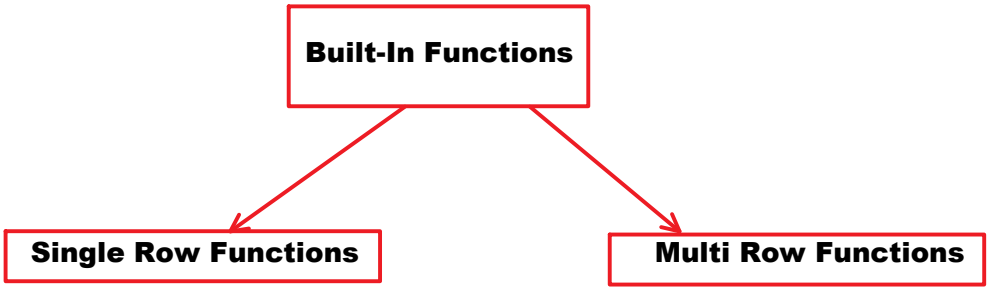
```
SELECT count(*) FROM emp
WHERE deptno=30;
```

Note:
to find number of records we can write following queries:

```
SELECT count(*) FROM emp;   --17

SELECT count(8) FROM emp;   --17

SELECT count(500) FROM emp;   --17
```

Differences b/w count(*) and count(<any_number>):

count(*)	• it counts no of records • it is slower
count(8)	• it counts number of 8s • it is faster



If 1 function call applied on 1 row

String Functions
Conversion Functions
Number Functions
.
.

If 1 function applied on multiple rows

Aggregate Functions

single row function

```
ENAME      LOWER(ENAME)
-----------  --------------------------
ALLEN      LOWER('ALLEN')
SMITH      LOWER('SMITH')
WARD       LOWER('WARD')
```

multi row function

```
SAL
-------      sum(Sal)
10000
20000        1 function call
5000         applied on
             3 rows
```

Number Functions / Math Functions:

power() Mod()

power()	Mod()
sqrt()	Ceil()
sign()	Floor()
abs()	Trunc()
	Round()

power():

- it is used to find power value.

Syntax:

power(<number>, <power>)

Examples:

power(2,3)	8
power(3,2)	9

sqrt():

it is used to find square root value.

Syntax:

sqrt(<number>)

Examples:

sqrt(100)	10
sqrt(25)	5

sign():

- it is used to check whether the number is +ve or -ve or 0.
- if num is +ve, returns 1
- if num is -ve, returns -1
- if num is 0, returns 0

Syntax:

sign(<number>)

Examples:

sign(25)	1
sign(-25)	-1
sign(0)	0

abs():

- **it is used to get absolute value.**
- **absolute value => non-negative**

Syntax:

abs(<number>)

Examples:

abs(25)	25
abs(-25)	25

Mod():

- **it is used to get remainder value.**

Syntax:

Mod(<number>, <divisor>)

Examples:

Mod(5,2)	1
Mod(10,7)	3

Ceil():

it is used to get round up value

Syntax:

Ceil(<number>)

Floor():

it is used to get round down value

Syntax:

Floor(<number>)

Examples:

Floor() Ceil()

456 => 456.789 => 457

Ceil(456.789)	457
Floor(456.789)	456

TRUNC():

- **it is used to remove decimal places.**

Syntax:

TRUNC(<number> [, <no_of_decimal_places>])

Examples:

TRUNC(123.6789)	123
TRUNC(123.6789,1)	123.6
TRUNC(123.6789,2)	123.67
TRUNC(123.6789,3)	123.678

2nd arg can be -ve.
if 2nd arg is -ve, it does not give decimal places

-1	rounds in 10s
-2	rounds in 100s
-3	rounds in 1000s

TRUNC(356.78934,-1)	350 and 360 350
TRUNC(356.78934,-2)	300 and 400 300
TRUNC(5678.4567,-3)	5000 and 6000 5000

Round():

- **It considers avrg.**
- **If value is avrg or above avrg, it gives upper value.**
- **if value is below avrg, it gives lower value.**

Syntax:

Round(<number> [, <no_of_decimal_places>])

Examples:

Round(123.6789)	123 and 124 avrg: 123.5 124
Trunc(123.6789)	123
Round(123.3789)	123 and 124 avrg: 123.5 123

Trunc(123.3789)	123
Round(123.5)	123 and 124 avrg: 123.5 124

Difference b/w trunc() and round():

trunc()	does not consider avrg always gives lower value
round()	considers avrg if value is avrg or above avrg, it gives upper value. if value is below avrg, it gives lower value

Round(456.6789,2)	456.68
Trunc(456.6789,2)	456.67
Round(456.6739,2)	456.67
Round(345.678923,3)	345.679
TRUNC(345.678923,3)	345.678

2nd arg can be -ve

120+130 = 250
250/2 = 125

Round(123.678,-1)	120 and 130 avrg: 125 120
Round(127.678,-1)	120 and 130 avrg: 125 130

Date Functions:

sysdate
systimestamp

Add_Months()
Months_Between()
Last_day()
Next_day()

sysdate:

- **it returns current system date**

  Syntax:

  sysdate

systimestamp:

- **it returns current system date and time**

  Syntax:

  systimestamp

Display current system date:

SELECT sysdate FROM dual;

Display current system time from sysdate:

SELECT to_char(sysdate, 'HH:MI:SS AM') FROM dual;

Display current system time from sysdate in 24hrs format:

SELECT to_char(sysdate, 'HH24:MI:SS') FROM dual;

Display current system date and time:

SELECT systimestamp FROM dual;

Add_Months():

- **it is used to add or subtract months to specific date or from specific date.**

  Syntax:

  Add_Months(<date>, <no_of_months>)

Add 2 days to today's date:

SELECT sysdate+2 FROM dual;

Add 2 months to today's date:

SELECT add_months(sysdate,2) FROM dual;

Add 2 years to today's date:

SELECT add_months(sysdate,2*12) FROM dual;

Subtract 2 days from today's date:

SELECT sysdate-2 FROM dual;

Subtract 2 months from today's date:

SELECT add_months(sysdate,-2) FROM dual;

Subtract 2 years from today's date:

SELECT add_months(sysdate,-2*12) FROM dual;

Examples:

Orders

Order_id	cid	pid	ordered_Date	delivery_Date
1234	..	..	sysdate	sysdate+5

Products

Pid	Pname	manufactured_date	expiry_date
5001	XYZ	sysdate	Add_Months(sysdate,3)

EMPLOYEE

EMPID	ENAME	DOBirth	DORetirement
1001	A	25-DEC-2000	Add_months(DOBirth, 60*12)

CMS_LIST

STATE_CODE	CM_NAME	START_DATE	END_DATE
TG	RR	7-DEC-23	Add_Months(start_Date, 5*12)

Examples:

```
INSERT INTO emp(empno,ename,hiredate)
VALUES(5001,'A',sysdate);
```

```
INSERT INTO emp(empno,ename,hiredate)
VALUES(5002,'B',sysdate-1);

INSERT INTO emp(empno,ename,hiredate)
VALUES(5003,'C',Add_Months(sysdate,-1));

INSERT INTO emp(empno,ename,hiredate)
VALUES(5004,'D',Add_Months(sysdate,-12));

COMMIT;
```

Display the emp records who joined today:

```
SELECT ename, hiredate
FROM emp
WHERE hiredate = sysdate;
        12-JUL-24 8.12 = 12-JUL-24 8.15   FALSE
```

Output:
no rows selected.

```
SELECT ename, hiredate
FROM emp
WHERE TRUNC(hiredate) = TRUNC(sysdate);
        TRUNC('12-JUL-24 8.12')        TRUNC('12-JUL-24 8.21')
        12-JUL-24                  =      12-JUL-24    TRUE
```

Output:
displays emp records who joined today

NOTE:
TRUNC() function can be used to remove time from date and time.

```
SQL> SELECT TRUNC(systimestamp) FROM dual;

TRUNC(SYS
---------
12-JUL-24
```

Display the emp records who joined yesterday:

```
SELECT ename, hiredate
FROM emp
WHERE TRUNC(hiredate) = TRUNC(sysdate-1);
```

Display the emp records who joined 1 month ago:

```
SELECT ename, hiredate
FROM emp
WHERE TRUNC(hiredate) = TRUNC(Add_Months(sysdate,-1));
```

Display the emp records who joined 1 year ago:

```
SELECT ename, hiredate
FROM emp
WHERE TRUNC(hiredate) = TRUNC(Add_Months(sysdate,-12));
```

Assignment:

GOLDRATES

dateid	price
1-JAN-2020	45000
2-JAN-2020	48000
..	
..	
12-JUL-24	68000

find today's goldrate:

WHERE trunc(dateid) = trunc(sysdate)

find yesterday's goldrate:

find 1 month ago goldrate:

find 1 year ago goldrate:

SALES

DATEID	AMOUNT
1-JAN-2020	90000
2-JAN-2020	85000
..	
..	
12-JUL-24	95000

find today's sales

find yesterday sales

find 1 month ago sales

find 1 year ago sales

Months_Between():

- **It is used to find difference between 2 dates.**

Syntax:

Months_Between(<date1>, <date2>)

Example:

Months_Between('12-JUL-24', '12-JUL-23')	12 [months]
Months_Between('12-JUL-24', '12-JUL-23')/12	1 [year]

Display emp names, hiredates and experience:

```
SELECT ename, hiredate,
TRUNC(Months_Between(sysdate, hiredate)/12) AS experience
FROM emp;
```

(or)

```
SELECT ename, hiredate,
TRUNC((sysdate-hiredate)/365) AS experience
FROM emp;
```

Display empnames, hiredates and experience.
display experience in the form of years and months:

15 months
30 months

years	months
TRUNC(15/12) = 1	Mod(15,12) = 3
TRUNC(30/12) = 2	Mod(30,12) = 6

```
SELECT ename, hiredate,
TRUNC(months_between(sysdate,hiredate)/12) AS years,
Mod(TRUNC(months_between(sysdate,hiredate)), 12) AS months
FROM emp;
```

Assignment:
Find age of SACHIN if DOB is: 24-APR-1973

TRUNC(Months_Between(sysdate, '24-APR-1973')/12)

Last_day():

- **it is used to get last date in the month.**

  Syntax:
  Last_day(<Date>)

  Examples:

Last_day(sysdate)	31-JUL-24
Last_day('17-FEB-2024')	29-FEB-24
Last_day('17-FEB-2023')	28-FEB-23

Find next month first date:

SELECT last_day(sysdate)+1 FROM dual;

31-JUL-24 + 1

1-AUG-24

Find current month first date:

SELECT Last_day(Add_Months(sysdate,-1))+1 FROM dual;

Last_day(12-JUN-24) => 30-JUN-24 + 1

1-JUL-24

Next_Day():

- **it is used to find next date based on weekday.**
- **For example, to find next Sunday date we use it.**

  Syntax:
  Next_day(<date>, <weekday_name>)

  Examples:

  Find next Sunday date:

  SELECT next_Day(sysdate, 'sun') FROM dual;

  Find next Friday date:

  SELECT next_Day(sysdate, 'fri') FROM dual;

Find next month first Sunday date:

```
SELECT Next_day(last_day(sysdate), 'sun')
FROM dual;
```

Find current month last Sunday date:

```
SELECT Next_day(last_day(sysdate)-7, 'sun')
FROM dual;
```

Analytic Functions / Window Functions:

Rank()
Dense_Rank()
Row_Number()

Rank() & Dense_Rank():

MARKS

670
950
730
840
950
840
500
730
400
730

ORDER BY marks DESC

MARKS	RANK	DENSE_RANK
950	1	1
950	1	1
840	3	2
840	3	2
730	5	3
730	5	3
730	5	3
670	8	4
500	9	5
400	10	6

Rank():

- **It is used to apply ranks to records according to**

specific column order.

Syntax:

```
Rank() OVER([PARTITION BY <column>]
ORDER BY <column> ASC/DESC)
```

Dense_Rank():

- It is used to apply ranks to records according to specific column order.

Syntax:

```
Dense_Rank() OVER([PARTITION BY <column>]
ORDER BY <column> ASC/DESC)
```

Examples on Rank() and Dense_Rank():

Display all emp names and salaries.
Apply ranks to the records according to salary.
give top rank to highest salary [apply ranks to records according to salary descending order]:

```
SELECT ename, sal,
Rank() OVER(ORDER BY sal DESC) AS rank
FROM emp;
```

(or)

```
SELECT ename, sal,
Dense_Rank() OVER(ORDER BY sal DESC) AS rank
FROM emp;
```

Display all emp names and hiredates.
Apply ranks to the records according to seniority.
Give top rank to most senior.

ORDER BY hiredate ASC

17-DEC-1982	15-AUG-1980	min date => max experience
15-AUG-1980	27-OCT-1981	
27-OCT-1981	17-DEC-1982	

```
SELECT ename, hiredate,
dense_rank() over(ORDER BY hiredate ASC) AS rank
FROM emp;
```

Display all emp records.
apply ranks to records according to salary descending order.
IF salary is same don't apply same rank. apply rank according to seniority:

5000	18-AUG-1990	1
3000	25-DEC-1981	2
3000	23-OCT-1982	3

```
SELECT ename, hiredate,
dense_rank() over(ORDER BY sal DESC, hiredate ASC) AS rank
FROM emp;
```

Apply ranks to emp records according to salary descending order with in dept:

DEPTNO	SAL
10	8000
10	10000
10	9000
20	15000
20	5000
20	7000
30	20000
30	10000
30	18000

DEPTNO	SAL	
10	10000	1
10	9000	2
10	8000	3
20	15000	1
20	7000	2
20	5000	3
30	20000	1
30	18000	2
30	10000	3

```
break on deptno skip 1 duplicates

SELECT ename, deptno, sal,
dense_rank() over(PARTITION BY deptno ORDER BY sal DESC) AS rank
FROM emp;

clear breaks
```

PARTITION BY clause:

- it is used to group the records according to specific column

ORDER BY clause:

- it is used to arrange the records in ascending or descending order according to specific column

Display all emp records. apply ranks to records according to seniority with in dept:

```
SELECT ename, deptno, hiredate,
dense_rank() over(partition by deptno order by hiredate asc) as rank
FROM emp;
```

Display all emp records. apply ranks to records according to salary descending order with in job:

```
SELECT ename, job, sal,
dense_rank() over(partition by job order by sal desc) AS rank
FROM emp;
```

Row_Number():

- it is used to apply row numbers to records

Syntax:

```
Row_Number(PARTITION BY <column>
ORDER BY <column> ASC/DESC)
```

Examples:

Display all emp records. apply row numbers according to empno ascending order:

```
SELECT row_number() OVER(ORDER BY empno ASC) AS sno,
empno, ename, sal
FROM emp;
```

Apply row numbers with in deptno according to salary desc:

```
SELECT row_number() over(partition by deptno order by sal desc as sno,
ename, deptno, sal
FROM emp;
```

Special Functions:

NVL()
NVL2()

Greatest()
Least()

Decode()

User
Uid

NVL():

- **It is used to replace the nulls.**
- **If first argument is not null, it returns first argument.**
- **If first argument is null, it returns 2nd argument.**

Syntax:

```
NVL(<arg1>, <arg2>)
```

Examples:

NVL(100,200)	100
NVL(null,200)	200

Examples on NVL():

Calculate Total salary of all emps [sal+comm]:

```
SELECT ename, sal, comm,
sal+NVL(comm,0) AS "Total Salary"
FROM emp;
```

Display all emp records along with commission.
If commission is null, replace it with N/A:

```
SELECT ename, sal, NVL(to_char(comm), 'N/A') AS comm
FROM emp;
```

Assignment:

STUDENT

SID	SNAME	M1
1001	A	60
1002	B	
1003	C	75
1004	D	

Displa all students records. if m1 marks are null, replace it with ABSENT

```
NVL(to_char(m1), 'ABSENT')
```

NVL2():

- **It is used to replace nulls and not nulls.**
- **If arg1 is not null, it returns arg2**
- **If arg1 is null, it returns arg3**

Syntax:

```
NVL2(<Arg1>, <arg2>, <arg3>)
```

Examples:

NVL2(100, 200, 300)	200
NVL2(null, 200, 300)	300

Example on NVL2():

Set all emps commission as following:

if emp is getting commission then increase 1000 rupees commission
if emp is not getting comm then set comm as 900

```
UPDATE emp
SET comm=NVL2(comm, comm+1000, 900);
```

Differences b/w NVL() and NVL2():

NVL()	• is used to replace the nulls. • it can take 2 arguments.
NVL2()	• is used to replace nulls and not nulls. • it can take 3 arguments.

Greatest():

- It is to find max value in horizontal values.

Syntax:

```
Greatest(<value1>, <v2>, <v3>, ........ , <value_n>)
```

Examples:

Greatest(10,20,30)	30
Greatest(6,5,8,7,3,2,4)	8

Find max value in each row:

```
SELECT greatest(f1, f2, f3) AS max_value
FROM t1;
```

T1

F1	F2	F3
45	78	60
74	32	81
30	55	77

```
greatest(f1, f2, f3)
-----------------------------
greatest(45, 78, 60)  => 78
greatest(74, 32, 81)  => 81
greatest(30, 55, 77)  => 77
```

Find max value in f3 column:

SELECT max(f3) FROM t1; --81

F3
60
81
77

max(f3)

Greatest()	• it is used to find max value in horizontal values • it is single row function • it can take variable length arguments
Max()	• it is used to find max value in vertical values • it is multi row function • it can take 1 argument

Least():

- it is used to find minimum value in horizontal values.
- single row function
- it can take variable length arguments

Min():

- it is used to find minimum value in vertical values.
- multi row function.
- it can take 1 argument

User:

- it returns current user name

Uid:

- it returns current user id

Example:

SELECT user, uid FROM dual;

show user
(or)
select user from dual;

Decode():

- **It is used to implement "IF .. THEN" in SQL.**
- **It can check equality condition only.**

Syntax:

```
Decode(<column>,
       <value1>, <expression1>,
       <value2>, <expression2>,
        .
        .
        [else_epression>)
```

Example on Decode():

Display all emp records along with job titles as following:

PRESIDENT => BIG BOSS
MANAGER => BOSS
Others => EMPLOYEE

```
SELECT ename,
decode(job,
       'PRESIDENT', 'BIG BOSS',
       'MANAGER', 'BOSS',
       'EMPLOYEE') AS job,
sal
FROM emp;
```

Increase salary of emps as following:
if deptno 10, increse 10% on sal
if deptno 20, increase 20% on sal
others, increase 15% on sal

```
UPDATE emp
SET sal=decode(deptno,
               10, sal+sal*0.1,
               20, sal+sal*0.2,
               sal+sal*0.15);
```

CLAUSES

Tuesday, July 16, 2024 8:05 AM

Clauses of SELECT command:

- **Clause is a part of query.**
- **Every query is made up of with clauses.**
- **Every clause has specific purpose.**

SQL
QUERIES
CLAUSES

ENGLISH
SENTENCES
WORDS

Syntax of SELECT command:

```
SELECT [ALL/DISTINCT] <columns_list>
FROM <tables_list>
WHERE <condition>
GROUP BY <grouping_columns_list>
HAVING <group_condition>
ORDER BY <column> ASC/DESC, <column> ASC/DESC, ..
OFFSET <number> ROW/ROWS
FETCH FIRST/NEXT <number> ROW/ROWS ONLY;
```

SELECT command clauses are:

- **SELECT**
- **FROM**
- **WHERE**
- **ORDER BY**
- **GROUP BY**
- **HAVING**
- **OFFSET**
- **FETCH**
- **DISTINCT**

SELECT:

- **it is used to specify columns list**

Syntax:

SELECT <columns_list>

Examples:

SELECT ename, sal
SELECT *

*	All columns

FROM:

- **It is used to specify tables list**
- **it selects entire table**

Syntax:

FROM <tables_list>

Examples:

FROM emp
FROM emp, dept

WHERE:

- **it is used to specify filter condition**
- **it filters the rows**

Syntax:

WHERE <condition>

Examples:

WHERE job='MANAGER'
WHERE deptno=30

Display the emp names and salaries whose salary is more than 3000:

```
SELECT ename, sal
FROM emp
WHERE sal>3000;
```

EMP

EMPNO	ENAME	SAL
1234	A	7000
1235	B	2500
1236	C	5000
1237	D	2000

FROM emp:
it selects entire emp table

EMP

EMPNO	ENAME	SAL
1234	A	7000
1235	B	2500
1236	C	5000
1237	D	2000

WHERE sal>3000:

- **it filters the rows**
- **WHERE condition will be applied on every row**

EMP

EMPNO	ENAME	SAL
1234	A	7000
1235	B	2500
1236	C	5000
1237	D	2000

WHERE sal>3000

7000>3000 T
2500>3000 F
5000>3000 T
2000>3000 F

EMPNO	ENAME	SAL
1234	A	7000

1236	C	5000

SELECT ename, sal:

- **it selects specified columns**

ENAME	SAL
A	7000
C	5000

ORDER BY:

- **It is used to arrange the records in ascending or descending order according to specific column(s).**
- **Default order is: ASC**

Syntax:

ORDER BY <column> ASC/DESC, <column> ASC/DESC, ...

Example:

NUMBER		CHAR		DATE	
ASC	DESC	ASC	DESC	ASC	DESC
1	10	A	Z	1-JAN-23	31-DEC-24
2	9	B	Y	2-JAN-23	30-DEC_24
3	8	.	.	.	.
.	.	.	.	.	.
.	.	Z	A	31-DEC-23	1-JAN-24
10	1			1-JAN-24	31-DEC-23
				.	.
				.	.
				31-DEC-24	1-JAN-23

Examples on ORDER BY:

Display all emp names and salaries.
Arrange emp names in Alphabetical order:

```
SELECT ename, sal
FROM emp
ORDER BY ename ASC;
```

(or)

```
SELECT ename, sal
FROM emp
ORDER BY ename;
```

(or)

```
SELECT ename, sal
FROM emp
ORDER BY 1 ASC;
```

ename	1
sal	2

(or)

```
SELECT *
FROM emp
ORDER BY 2 ASC;
```

*	empno, ename, job, mgr, hiredate, sal, comm, deptno

Display all emp names and salaries.
arrange salaries in descending order:

```
SELECT ename, sal
FROM emp
```

```
ORDER BY sal DESC;
```

(or)

```
SELECT ename, sal
FROM emp
ORDER BY 2 DESC;
```

Display all emp records.
Arrange them in ascending order according to empno:

```
SELECT *
FROM emp
ORDER BY empno ASC;
```

Display all emp records.
Arrange them in ascending order according to deptno:

```
break on deptno skip 1 duplicates

SELECT ename, sal, deptno
FROM emp
ORDER BY deptno ASC;

clear breaks
```

Display all emp records according to seniority:

```
SELECT ename, hiredate
FROM emp
ORDER BY hiredate ASC;
```

min date => max experience

Arranging records in order according to multiple columns:

Display all emp records.
Arrange them in ascending order according to deptno.
Within the dept arrange salaries in descending order:

```
SELECT ename, deptno, sal
FROM emp
ORDER BY deptno ASC, sal DESC;
```

ORDER BY deptno ASC, sal DESC

if deptno is different, it will not check salary.
if deptno is same, then only it checks with salary.

CASE-1: deptno is different

20	10
10	20

CASE-1: deptno is same

10	4000	10	6000
10	6000	10	4000

Display all emp records.
Arrange them according to seniority with in dept:

```
SELECT ename, deptno, hiredate
FROM emp
ORDER BY deptno ASC, hiredate ASC;
```

Display all emp records.
Arrange them in ascending order according to deptno.

Within dept arrange salaries in descending order.
If salary is same arrange them according to seniority:

```
SELECT ename, deptno, sal, hiredate
FROM emp
ORDER BY deptno ASC, sal DESC, hiredate ASC;
```

Note:
In ASCENDING ORDER, nulls will be displayed last.
In DESCENDING ORDER, nulls will be displayed first.

Display all emp records. Arrange them in descending order according to salary. Display nulls last:

```
SELECT ename, sal
FROM emp
ORDER BY sal DESC NULLS LAST;
```

Display all emp records. Arrange them in ascending order according to salary. Display nulls first:

```
SELECT ename, sal
FROM emp
ORDER BY sal ASC NULLS FIRST;
```

GROUP BY:

- **It is used to group the records according to specific column(s).**
- **On these groups we can apply aggregate functions.**
- **It gives summarized data from detailed data.**
- **It can be used for data analysis.**

Syntax:

```
GROUP BY <columns_list>
```

Example:

GROUP BY deptno => 10 group, 20 group

GROUP BY job => MANAGER group, CLERK group

EMP => detailed data → **GROUP BY deptno** → **summarized data**

EMPNO	ENAME	DEPTNO	SAL
1001	A	30	10000
1002	B	30	5000
1003	C	10	20000
1004	D	10	10000
1005	E	20	12000
1006	F	20	8000

deptno	sum_of_Sal
10	30000
20	20000
30	15000

Examples on GROUP BY:

Find dept wise sum of salaries:

DEPTNO	SUM_OF_SAL
10	?
20	?
30	?

```
SELECT deptno, sum(sal) AS sum_of_sal
FROM emp
GROUP BY deptno
ORDER BY 1;
```

Find dept wise no of emps:

DEPTNO	NO_OF_EMPS
10	?
20	?
30	?

```
SELECT deptno, count(*) AS no_of_emps
FROM emp
GROUP BY deptno
ORDER BY 1;
```

Find dept wise max salary and min salary:

DEPTNO	MAX_SAL	MIN_SAL
10	?	?
20	?	?
30	?	?

```
SELECT deptno, max(sal) AS max_sal, min(sal) AS min_sal
FROM emp
GROUP BY deptno
ORDER BY 1;
```

Find dept wise avrg salary:

DEPTNO	AVG_SAL
10	?
20	?

```
SELECT deptno, avg(sal) AS avg_sal
FROM emp
GROUP BY deptno
ORDER By 1;
```

Find year wise no of emps joined in organization:

YEAR	NO_OF_EMPS
1980	?
1981	?
1982	?

```
SELECT to_char(hiredate, 'YYYY') AS year,
count(*) AS no_of_emps
FROM emp
GROUP BY to_char(hiredate, 'YYYY')
ORDER BY 1;
```

Find quarter wise no of emps joined in organization:

QUARTER	NO_OF_EMPS
1	?
2	?
3	?
4	?

```
SELECT to_char(hiredate,'Q') AS quarter,
count(*) AS no_of_emps
FROM emp
GROUP BY to_char(hiredate,'Q')
ORDER BY 1;
```

Assignment:
Find week day wise no of emps joined in org

WEEKDAY	NO_OF_EMPS
1	?

2	?
3	?

Find job wise sum of salaries:

JOB	SUM_OF_SAL
MANAGER	?
CLERK	?
SALESMAN	?

```
SELECT job, sum(Sal) AS sum_of_sal
FROM emp
GROUP BY job;
```

Assignment:

Find job wise no of emps

JOB	NO_OF_EMPS
CLERK	?
MANAGER	?

Find job wise max sal and min sal:

JOB	MAX_SAL	MIN_SAL
CLERK	?	?
SALESMAN	?	?

Find job wise avrg sal

JOB	AVG_SAL
CLERK	?
SALESMAN	?

PERSON

PID	PNAME	STATE	GENDER	AADHAR
		TG	M	
		TG	F	
		TG	F	
		TG	M	
		AP	F	
		AP	F	
		AP	M	
		AP	M	

Find state wise population:

state	population
TG	?
AP	?

```
SELECT state, count(*) AS population
FROM person
GROUP BY state
ORDER BY state;
```

Find gender wise population:

gender	population
M	?
F	?

```
SELECT gender, count(*) AS population
FROM person
GROUP BY gender;
```

Grouping records according to multiple columns:

Find dept wise, job wise sum of salaries:

DEPTNO	JOB	SUM_OF_SAL
10	CLERK	?
10	MANAGER	?
20	CLERK	?
20	MANAGER	?

```
SELECT deptno, job, sum(sal) AS sum_of_sal
FROM emp
GROUP BY deptno, job
ORDER BY 1;
```

ROLLUP() and CUBE():

These are used to calculate sub totals and grand total.

Rollup():

It calculates sub totals and grand total according to first column in GROUP BY columns list

Syntax:

GROUP BY ROLLUP(<grouping_columns_list>)

Example:

GROUP BY Rollup(deptno, job)

it calculates sub totals and grand total according to deptno

Cube():

It calculates sub totals and grand total according to all columns in GROUP BY columns list

Syntax:

GROUP BY CUBE(<grouping_columns_list>)

Example:

GROUP BY Cube(deptno, job)

Example on ROLLUP() and CUBE():

Find dept wise, job wise sum of salaries.
Find sub totals and grand total according to deptno
[Rollup()]:

DEPTNO	JOB	SUM_OF_SAL
10	CLERK	?
	MANAGER	?
	10th dept sub total	?
20	CLERK	?
	MANAGER	?
	20th dept sub total	?
	GRAND TOTAL	?

```
SELECT deptno, job, sum(Sal) AS sum_of_sal
FROM emp
GROUP BY Rollup(deptno, job)
ORDER BY 1;
```

Find dept wise, job wise sum of salaries.
Find sub totals and grand total according to deptno and job
[Cube()]:

DEPTNO	JOB	SUM_OF_SAL
10	CLERK	?
	MANAGER	?
	10th dept sub total	?
20	CLERK	?

	MANAGER	?
	20th dept sub total	?
	CLERK sub total	?
	MANAGER sub total	?
	GRAND TOTAL	?

```
SELECT deptno, job, sum(Sal) AS sum_of_sal
FROM emp
GROUP BY Cube(deptno, job)
ORDER BY 1;
```

Find year wise, quarter wise no of emps joined in organization:

YEAR	QUARTER	NO_OF_EMPS
1980	1	?
	2	?
	3	?
	4	?
1981	1	?
	2	?
	3	?
	4	?

```
SELECT to_char(hiredate,'YYYY') AS year,
to_char(hiredate, 'Q') AS quarter,
count(*) AS no_of_emps
FROM emp
GROUP BY to_char(hiredate,'YYYY'), to_char(hiredate, 'Q')
ORDER BY 1;
```

Find year wise, quarter wise no of emps joined in organization. Calculate sub totals and gran total according to year [Rollup()]

YEAR	QUARTER	NO_OF_EMPS
1980	1	?
	2	?
	3	?
	4	?
	1980 sub total	?
1981	1	?
	2	?
	3	?
	4	?
	1981 sub total	?
	Grand total	?

```
SELECT to_char(hiredate,'YYYY') AS year,
to_char(hiredate, 'Q') AS quarter,
count(*) AS no_of_emps
FROM emp
GROUP BY Rollup(to_char(hiredate,'YYYY'), to_char(hiredate, 'Q'))
ORDER BY 1;
```

Find year wise, quarter wise no of emps joined in organization. Calculate sub totals and gran total according to year and quarter [Cube()]

YEAR	QUARTER	NO_OF_EMPS
1980	1	?
	2	?
	3	?
	4	?
	1980 sub total	?
1981	1	?
	2	?
	3	?

	4	?
	1981 sub total	?
	1st quarter sub total	?
	2nd quarter sub total	?
	3rd quarter sub total	?
	4th quarter sub total	?
	Grand total	?

```
SELECT to_char(hiredate,'YYYY') AS year,
to_char(hiredate, 'Q') AS quarter,
count(*) AS no_of_emps
FROM emp
GROUP BY Cube(to_char(hiredate,'YYYY'), to_char(hiredate, 'Q'))
ORDER BY 1;
```

Assignment:

SALES

DATEID	AMOUNT
1-JAN-2022	80000
2-JAN-2022	70000
..	
..	
18-JUL-2024	90000

Find year wise, quarter wise sales
calculate sub totals and grand total according to year and quarter:

```
Sum (Amount)
GROUP BY CUBE(to_char(dateid, 'YYYY'), to_char(dateid, 'Q'))
```

YEAR	QUARTER	SALES
2022	1	?
	2	?

	3	?
	4	?
	2022 sub total	?
2023	1	?
	2	?
	3	?
	4	?
	2023 sub total	?
	1st quarter sub total	?
	2nd sub total	?
	3rd	?
	4th	?
	GRAND TOTAL	?

Assignment:

PERSON

PID	PNAME	STATE	GENDER	AADHAR
		TG	M	
		TG	F	
		TG	F	
		TG	M	
		AP	F	
		AP	F	
		AP	M	
		AP	M	

- **Find state wise, gender wise population**
  GROUP BY state, gender

- **Find state wise, gender wise population. calculate state wise population [state wise sub total] and gender wise population [gender wise sub total] and INDIA population [grand total]:**

GROUP BY Cube(state, gender)

WHERE <condition> => rows
HAVING <condition> => groups

HAVING:

- **HAVING clause is used to write conditions on groups.**
- **It will be applied on result of GROUP BY.**
- **It cannot be used without GROUP BY.**
- **It filters the groups.**

Syntax:

```
HAVING <group_condition>
```

Examples on HAVING:

Display the depts which are spending more than 10000 rupees amount on their emps:

```
SELECT deptno, sum(sal)
FROM emp
GROUP BY deptno
HAVING sum(sal)>10000;
```

Display the depts which are having 5 or more emps:

```
SELECT deptno, count(*)
FROM emp
GROUP BY deptno
HAVING count(*)>=5;
```

Differences b/w WHERE and HAVING:

WHERE	HAVING
• it is used to write conditions on rows.	• it is used to write conditions on groups.
• it filters the rows.	• it filters the groups.
• it can be used without GROUP BY.	• it cannot be used without GROUP BY.
• it gets executed before GROUP BY	• it gets executed after GROUP BY
• we cannot use aggregate function in WHERE clause.	• we can use aggregate function in HAVING clause

Execution Order [oracle 21c]:

FROM
WHERE
GROUP BY
HAVING
SELECT
DISTINCT
ORDER BY
OFFSET
FETCH

OFFSET:

- **introduced in ORACLE 12C.**

- **it is used to specify no of rows to be skipped.**

  Syntax:

  OFFSET <number> ROW/ROWS

FETCH:

- **introduced in ORACLE 12C.**
- **it is used to specify no of rows to be fetched.**

  Syntax:

  FETCH <FIRST/NEXT> <number> ROW/ROWS ONLY

Examples on OFFSET and FETCH:

Display all emp records except first 5 rows:

```
SELECT * FROM emp
OFFSET 5 ROWS;
```

Display first 5 rows only from emp table:

```
SELECT * FROM emp
FETCH FIRST 5 ROWS ONLY;
```

Display 6th row to 10th row:

```
SELECT * FROM emp
OFFSET 5 ROWS
FETCH NEXT 5 ROWS ONLY;
```

Display top 3 salaried emp records:

```
SELECT ename, sal
FROM emp
ORDER BY sal DESC
FETCH FIRST 3 ROWS ONLY;
```

Display top 3 seniors records:

```
SELECT ename, hiredate
```

```
FROM emp
ORDER BY hiredate ASC
FETCH FIRST 3 ROWS ONLY;
```

DISTINCT:

- **it is used to eliminate duplicate records.**

Syntax:

```
SELECT ALL/DISTINCT <columns_list>
```

Examples on DISTINCT:

Display the job titles offered by company:

```
SELECT job FROM emp;
(or)
SELECT ALL job FROM emp;

JOB
--------
CLERK
SALESMAN
MANAGER
SALESMAN
MANAGER
CLERK
CLERK
MANAGER
MANAGER
SALESMAN
```

```
SELECT DISTINCT job FROM emp;

JOB
-----------
CLERK
SALESMAN
MANAGER
```

Display the depts which are having emps:

```
SELECT deptno FROM emp;
(or)
SELECT ALL deptno FROM emp;

DEPTNO
```

```
SELECT DISTINCT deptno FROM emp
ORDER BY deptno;
```

```
SELECT ALL deptno FROM emp;

DEPTNO
-------------
20
30
30
10
20
10
10
20
30
```

```
SELECT DISTINCT deptno FROM emp
ORDER BY deptno;

DEPTNO
------------
10
20
30
```

Display dept wise job titles offered by company:

```
break on deptno skip 1 duplicates

SELECT DISTINCT deptno, job
FROM emp
ORDER BY deptno;
```

Execution Order of clauses of SELECT command:

FROM	to specify table names	FROM emp FROM emp, dept
WHERE	to specify filter condition it filters the rows it will be applied on every row	WHERE sal>3000 WHERE job='MANAGER'
GROUP BY	used to group the records according to specific column(s)	GROUP BY deptno GROUP BY job GROUP BY deptno, job

HAVING	used to write conditions on groups	HAVING sum(sal)>10000 HAVING count(*)>=5
SELECT	used to specify columns list	SELECT ename, sal SELECT *
DISTINCT	it eliminates the duplicates	SELECT DISTINCT job
ORDER BY	it is used to arrange the records in ASC or DESC order	ORDER BY sal DESC ORDER BY ename ASC
OFFSET	used to skip the rows	OFFSET 5 ROWS
FETCH	used to fetch the rows	FETCH FIRST 5 ROWS ONLY

Can we use column alias in GROUP BY?
NO. Because, GROUP BY gets executed before SELECT.
But, from ORACLE 23ai, we can use column alias in GROUP BY.

Can we use column alias in ORDER BY?
YES. Because, ORDER By gets executed after SELECT.

YEAR	NO_OF_EMPS
1980	?
1981	?

```
SELECT to_char(hiredate,'YYYY') AS year, count(*) AS no_of_emps
FROM emp
GROUP BY year
ORDER BY year;
```

Output:
ERROR: YEAR invalid IDENTIFIER

Execution Order [oracle 21c]:

FROM
WHERE
GROUP BY
HAVING
SELECT
DISTINCT
ORDER BY
OFFSET
FETCH

Till oracle 21c,
we cannot use column alias in GROUP BY, HAVING and WHERE.

In oracle 23AI,
we can use column alias in GROUP BY and HAVING

NOTE:

- **When we use group function, SELECT clause allows either GROUP BY column or GROUP FUNCTION.**

- **When we use GROUP BY, SELECT clause allows either GROUP BY column or GROUP FUNCTION.**

```
SELECT ename, max(sal) FROM emp;
Output:
ERROR

SELECT min(sal), max(sal) FROM emp;
--finds max sal and min sal


SELECT deptno, ename FROM emp
GROUP BY deptno;
Output:
ERROR
```

10

20
30

JOINS

Saturday, July 20, 2024 7:43 AM

JOINS:
GOAL:

- **It is used to retrieve the data from multiple tables**

COLLEGE DB

COURSE
STUDENT
MARKS
FEE

S.SID = M.SID

STUDENT S

SID	SNAME
1001	A
1002	B
1003	C
1004	D

MARKS M

SID	Maths	Phy	Che
1001	70	50	60
1002	55	44	81
1003	39	50	47

JOINS

SID	SNAME	MATHS

STUDENT MARKS

SORTING operation => arranging in ASC or DESC
FILTERING operation => filters the rows => sal>3000
JOIN operation => one table row combines [joins] with another table row

JOINS:

- **JOIN => combine / connect / link**
- **JOIN is an operation.**
- **In Join operation, one table record will be combined [joined] with another table record based on some condition. This condition is called "Join Condition". This operation is called Join operation**
- **JOIN CONDITION decides which record in one table should be joined with which record in another table.**
- **JOINS concept is used to retrieve the data from multiple tables.**

Types of Joins:

- **Inner Join => matched records only**
  - **Equi Join**

- Non-Equi Join
- **Outer Join** => matched + unmatched
  - Left Outer Join
  - Right Outer Join
  - Full Outer Join
- Self Join
- Cross Join / cartesian Join

Inner Join:

- Inner Join can give matched records only.
- It has 2 sub types. they are:
  - Equi Join
  - Non-Equi Join

Equi Join:

- If Join Operation is performed based on equality condition then it is called "Equi Join".

Examples on Equi Join:

Display student details along with maths subject marks:

SID	SNAME	MATHS

student.sid = marks.sid

STUDENT S

SID	SNAME
1001	A
1002	B
1003	C
1004	D

MARKS M

SID	Maths	Phy	Che
1001	70	50	60
1002	55	44	81
1003	39	50	47

```
CREATE TABLE student
(
sid NUMBER(4),
sname VARCHAR2(10)
);

INSERT INTO student VALUES(1001,'A');
INSERT INTO student VALUES(1002,'B');
INSERT INTO student VALUES(1003,'C');
INSERT INTO student VALUES(1004,'D');
COMMIT;


CREATE TABLE marks
(
sid NUMBER(4),
maths NUMBER(3),
phy NUMBER(3),
che NUMBER(3)
);
```

1001	70	50	60
1002	55	44	81
1003	39	50	47

```
INSERT INTO marks VALUES(1001,70,50,60);
INSERT INTO marks VALUES(1002,55,44,81);
INSERT INTO marks VALUES(1003,39,50,47);
COMMIT;
```

SID	SNAME	MATHS

STUDENT MARKS

```
SELECT student.sid, sname, maths
FROM student, marks
WHERE student.sid = marks.sid;
```

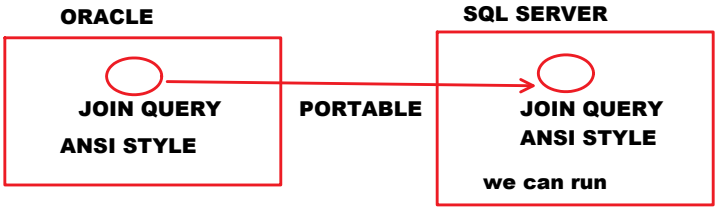
```
SELECT s.sid, sname, maths
FROM student s, marks m
WHERE s.sid = m.sid;
```

Above query degrades the performance

```
SELECT s.sid, s.sname, m.maths
FROM student s, marks m
WHERE s.sid = m.sid;
```

Note:
From ORACLE 9i version onwards, we can write JOIN QUERY in 2 styles. They are:

- ORACLE STYLE / NATIVE STYLE
- ANSI STYLE => best way [portable]



SID	SNAME	MATHS

STUDENT s MARKS m

ORACLE STYLE:

```
SELECT s.sid, s.sname, m.maths
FROM student s, marks m
WHERE s.sid=m.sid;
```

ANSI STYLE:

```
SELECT s.sid, s.sname, m.maths
FROM student s INNER JOIN marks m
ON s.sid=m.sid;
```

e.deptno = d.deptno

EMP e

EMPNO	ENAME	SAL	DEPTNO

DEPT d

DEPTNO	DNAME	LOC

EMP e

EMPNO	ENAME	SAL	DEPTNO
7369	SMITH	800	20
7499	ALLEN	1600	30
7521	WARD	1250	30
7566	JONES	2975	20
7782	CLARK	2450	10
7934	MILLER	1300	10
1001	A	1800	
1002	B	2000	

DEPT d

DEPTNO	DNAME	LOC
10	ACCOUNTS	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

Display emp details along with dept details:

ENAME	SAL	DNAME	LOC

EMP e DEPT d

ORACLE STYLE:

```
SELECT e.ename, e.sal, d.dname, d.loc
FROM emp e, dept d
WHERE e.deptno = d.deptno;
```

ANSI STYLE:

```
SELECT e.ename, e.sal, d.dname, d.loc
FROM emp e INNER JOIN dept d
ON e.deptno = d.deptno;
```

Display the emp records who are working in NEW YOEK

ename	sal	dname	loc
			NEW YORK

EMP e DEPT d

ORACLE STYLE:

```
SELECT e.ename, e.sal, d.dname, d.loc
FROM emp e, dept d
WHERE e.deptno=d.deptno AND d.loc='NEW YORK';
```

ANSI STYLE:

```
SELECT e.ename, e.sal, d.dname, d.loc
FROM emp e INNER JOIN dept d
ON e.deptno=d.deptno
WHERE d.loc='NEW YORK';
```

NOTE:

- **ON clause is used to specify JOIN CONDITION.**
- **WHERE clause is used to specify FILTER CONDITION.**

- **First, filter operation will be performed.. Then Join operation will be performed.**

e.deptno = d.deptno

EMP e

EMPNO	ENAME	SAL	DEPTNO
7369	SMITH	800	20
7499	ALLEN	1600	30
7521	WARD	1250	30
7566	JONES	2975	20

DEPT d

DEPTNO	DNAME	LOC
10	ACCOUNTS	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

7499	ALLEN	1600	30
7521	WARD	1250	30
7566	JONES	2975	20
7782	CLARK	2450	10
7934	MILLER	1300	10
1001	A	1800	
1002	B	2000	

20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

Note:
to see execution plan write following command:

```
SQL> SET AUTOTRACE ON EXPAIN

SQL> SET PAGES 200
```

Display ALLEN record with dept details:

ENAME	DNAME	LOC
ALLEN		

EMP e DEPT d

ORACLE STYLE:

```
SELECT e.ename, d.dname, d.loc
FROM emp e, dept d
WHERE e.deptno=d.deptno AND e.ename='ALLEN';
```

ANSI STYLE:

```
SELECT e.ename, d.dname, d.loc
FROM emp e INNER JOIN dept d
ON e.deptno=d.deptno
WHERE e.ename='ALLEN';
```

e,deptno=d.deptno

EMP e

EMPNO	ENAME	SAL	DEPTNO
7369	SMITH	800	20
7499	ALLEN	1600	30
7521	WARD	1250	30
7566	JONES	2975	20
7782	CLARK	2450	10
7934	MILLER	1300	10
1001	A	1800	

DEPT d

DEPTNO	DNAME	LOC
10	ACCOUNTS	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

Display the emp records who are working in SALES dept:

ENAME	DNAME

EMP e DEPT d

ORACLE STYLE:

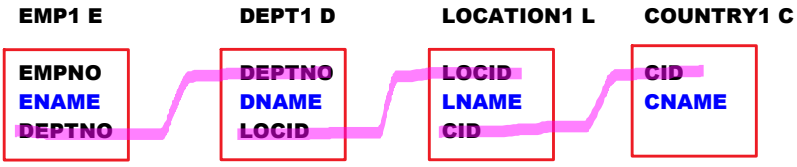
```
SELECT e.ename, d.dname
FROM emp e, dept d
WHERE e.deptno=d.deptno AND d.dname='SALES';
```

ANSI STYLE:

```
SELECT e.ename, d.dname
FROM emp e INNER JOIN dept d
ON e.deptno=d.deptno
WHERE d.dname='SALES';
```

Example on retrieving data from 4 tables:

e.deptno=d.deptno d.locid=l.locid l.cid=c.cid



ENAME	DNAME	LNAME	CNAME
EMP1 E	DEPT1 D	LOCATION1 L	COUNTRY1 C

```
create table emp1
(
empno number(4),
ename varchar2(10),
deptno number(2)
);

insert into emp1 values(1001,'A',10);

create table dept1
(
deptno number(2),
dname varchar2(10),
locid number(5)
);

insert into dept1 values(10,'SALES',12345);

create table location1
(
locid number(5),
lname varchar2(10),
cid number(3)
);

insert into location1 values(12345,'HYD',100);

create table country1
(
cid number(4),
cname varchar2(10)
);

insert into country1 values(100,'INDIA');

commit;
```

ENAME	DNAME	LNAME	CNAME

emp1 e dept1 d location1 l country1 c

ORACLE STYLE:

```
SELECT e.ename, d.dname, l.lname, c.cname
FROM emp1 e, dept1 d, location1 l, country1 c
WHERE e.deptno=d.deptno AND d.locid=l.locid
AND l.cid=c.cid;
```

ANSI STYLE:

```
SELECT e.ename, d.dname, l.lname, c.cname
FROM emp1 e INNER JOIN dept1 d
ON e.deptno=d.deptno INNER JOIN location1 l
ON d.locid=l.locid INNER JOIN country1 c
ON l.cid=c.cid;
```

Inner Join:
- Inner Join = matched records only
- Inner Join can give matched records only.

Outer Join:
- Outer Join = matched + unmatched records
- Inner Join can give matched records only. To get unmatched records also we use OUTER JOIN

It has 3 sub types. They are:
- Left Outer Join
- Right Outer Join
- Full Outer Join

Note:
- In ORACLE STYLE, based on join condition we can decide left table and right table.
- LHS table => left table
- RHS table => right table

Examples:
WHERE e.deptno = d.deptno

e	emp	left table
d	dept	right table

Examples:
WHERE d.deptno = e.deptno

d	dept	left table
e	emp	right table

In ANSI STYLE, based on keyword we can decide left table and right table.

Example:
FROM emp e INNER JOIN dept d

e	emp	left table

d	dept	right table

FROM dept d INNER JOIN emp e

d	dept	left table
e	emp	right table

Left Outer Join:

- Left Outer Join = matched + unmatched from left table
- left Outer Join can give matched records and unmatched records from left table.
- Outer Join Operator symbol: (+)
- In ORACLE STYLE,
  For Left Outer Join we write (+) symbol at right side.
- In ANSI STYLE,
  For Left Outer Join use the keyword: LEFT [OUTER] JOIN

Example on Left Outer Join:

```
INSERT INTO emp(empno,ename,sal)
VALUES(1001,'A',6000);

INSERT INTO emp(empno,ename,sal)
VALUES(1002,'B',4000);

COMMIT;
```

Display the emp records along with dept details.
Also display the emps to whom dept is not assigned:

ENAME	SAL	DNAME	LOC

EMP e DEPT d

ORACLE STYLE:

```
SELECT e.ename, e.sal, d.dname, d.loc
FROM emp e, dept d
WHERE e.deptno=d.deptno(+);
```

ANSI STYLE:

```
SELECT e.ename, e.sal, d.dname, d.loc
FROM emp e LEFT OUTER JOIN dept d
ON e.deptno=d.deptno;
```

Right outer join:

- Right Outer join = matched + unmatched from right table
- Right Outer join can give matched records and unmatched records from right table.
- In ORACLE STYLE, we write (+) symbol at left side.

- **In ANSI STYLE, use the keyword: RIGHT [OUTER] JOIN**

Example on Right Outer Join:

Display emp details along with dept details.
Also display the depts which are not having emps:

ENAME	SAL	DNAME	LOC

ORACLE STYLE:

```
SELECT e.ename, e.sal, d.dname, d.loc
FROM emp e, dept d
WHERE e.deptno(+)=d.deptno;
```

ANSI STYLE:

```
SELECT e.ename, e.sal, d.dname, d.loc
FROM emp e RIGHT OUTER JOIN dept d
ON e.deptno=d.deptno;
```

Full Outer Join:

- **Full Outer Join = matched + unmatched from left and right tables**
- **Full Outer Join can give matched records, unmatched records from left and right tables.**
- **In Oracle style, we use UNION operator**
- **In ANSI style, we use keyword: FULL [OUTER] JOIN**

A = {1,2,3,4,5}
B= {4,5,6,7,8}
A U B = {1,2,3,4,5,6,7,8}

In SQL:

Left Outer Join = matched + um from left
UNION
Right Outer Join = matched + um from right

Full Outer Join = matched + um from left + um from right

Example on Full Outer join:

Display emp details along with dept details.
Also display the emps to whom dept is not assigned.
Also display the depts in which emps are not existed.

ENAME	SAL	DNAME	LOC

ORACLE STYLE:

```
SELECT e.ename, e.sal, d.dname, d.loc
FROM emp e, dept d
WHERE e.deptno=d.deptno(+)
UNION
SELECT e.ename, e.sal, d.dname, d.loc
FROM emp e, dept d
WHERE e.deptno(+)=d.deptno;
```

ANSI STYLE:

```
SELECT e.ename, e.sal, d.dname, d.loc
FROM emp e FULL OUTER JOIN dept d
ON e.deptno=d.deptno;
```

Displaying Unmatched Records only:

Left Outer Join +Condition:

Left Outer Join + Condition = unmatched from left table

Example:
Display the emps to whom dept is not assigned.

ENAME	SAL	DNAME	LOC

EMP e DEPT d

ORACLE STYLE:

```
SELECT e.ename, e.sal, d.dname, d.loc
FROM emp e, dept d
WHERE e.deptno=d.deptno(+) AND d.dname IS null;
```

ANSI STYLE:

```
SELECT e.ename, e.sal, d.dname, d.loc
FROM emp e LEFT JOIN dept d
ON e.deptno=d.deptno
WHERE d.dname IS null;
```

Right Outer Join + Condition:

- **Right outer Join+condn = unmatched from right table**

  Example:
  Display the depts which are not having emps.

ENAME	SAL	DNAME	LOC

ORACLE STYLE:

```
SELECT e.ename, e.sal, d.dname, d.loc
FROM emp e, dept d
WHERE e.deptno(+) = d.deptno AND e.ename IS null;
```

ANSI STYLE:

```
SELECT e.ename, e.sal, d.dname, d.loc
FROM emp e RIGHT JOIN dept d
ON e.deptno = d.deptno
WHERE e.ename IS null;
```

Full Outer Join + Conditions:

- **Full Outer join + condns = um from left and right tables**

  Example:

  Display the emp records to whom dept is not assigned.
  Also display the depts in which emps are not existed:

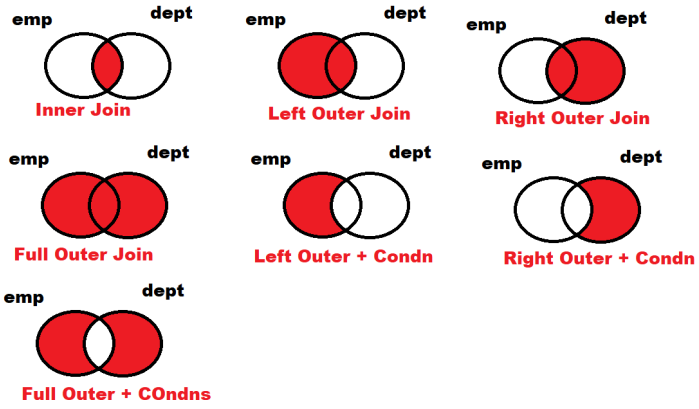
ORACLE STYLE:

```
SELECT e.ename, e.sal, d.dname, d.loc
FROM emp e, dept d
WHERE e.deptno=d.deptno(+) AND d.dname IS null
UNION
SELECT e.ename, e.sal, d.dname, d.loc
FROM emp e, dept d
WHERE e.deptno(+) = d.deptno AND e.ename IS null;
```

ANSI STYLE:

```
SELECT e.ename, e.sal, d.dname, d.loc
FROM emp e FULL JOIN dept d
ON e.deptno=d.deptno
WHERE d.dname IS null OR e.ename IS null;
```

Venn Diagrams of Joins:



Non-Equi Join:

- **If Join Operation is performed based on other than equality condition then it is called "Non-Equi Join".**

Examples:

```
WHERE e.deptno>d.deptno
WHERE e.deptno<d.deptno
WHERE e.deptno!=d.deptno
```

Example on Non-Equi Join:

e.sal BETWEEN s.losal AND s.hisal

EMP e

EMPNO	ENAME	SAL
1001	A	1500
1002	B	5000
1003	C	1300
1004	D	1000
1005	E	

SALGRADE s

GRADE	LOSAL	HISAL
1	700	1200
2	1201	1400
3	1401	2000
4	2001	3000
5	3001	9999

Display emp details with salary grades:

ENAME	SAL	GRADE

EMP e **SALGRADE s**

ORACLE STYLE:

```
SELECT e.ename, e.sal, s.grade
FROM emp e, salgrade s
WHERE e.sal BETWEEN s.losal AND s.hisal;
```

ANSI STYLE:

```
SELECT e.ename, e.sal, s.grade
FROM emp e INNER JOIN salgrade s
ON e.sal BETWEEN s.losal AND s.hisal;
```

Self Join / Recursive Join:

- **If a table is joined to itself then it is called "Self Join".**
- **In this, one table record will be joined with another record in same table.**

Example:

e.mgr = m.empno

EMP e

EMPNO	ENAME	JOB	SAL	MGR
1001	A	MANAGER	20000	
1002	B	CLERK	8000	1001
1003	C	ANALYST	6000	1001
1004	D	MANAGER	25000	
1005	E	CLERK	9000	1004

EMP m

EMPNO	ENAME	JOB	SAL	MGR
1001	A	MANAGER	20000	
1002	B	CLERK	8000	1001
1003	C	ANALYST	6000	1001
1004	D	MANAGER	25000	
1005	E	CLERK	9000	1004

Display emp details along with managers details:

EMP_NAME	EMP_SAL	MGR_NAME	MGR_SAL

ORACLE STYLE:

```
SELECT e.ename AS emp_name, e.sal AS emp_sal,
m.ename AS mgr_name, m.sal AS mgr_sal
FROM emp e, emp m
WHERE e.mgr = m.empno;
```

ANSI STYLE:

```
SELECT e.ename AS emp_name, e.sal AS emp_sal,
m.ename AS mgr_name, m.sal AS mgr_sal
FROM emp e INNER JOIN  emp m
ON e.mgr = m.empno;
```

Display the emp records who are earning more than their manager:

EMP_NAME	EMP_SAL	MGR_NAME	MGR_SAL

ORACLE STYLE:

```
SELECT e.ename AS emp_name, e.sal AS emp_sal,
m.ename AS mgr_name, m.sal AS mgr_Sal
FROM emp e, emp m
WHERE e.mgr=m.empno AND e.sal>m.sal;
```

ANSISTYLE:

```
SELECT e.ename AS emp_name, e.sal AS emp_sal,
m.ename AS mgr_name, m.sal AS mgr_Sal
FROM emp e INNER JOIN emp m
ON e.mgr=m.empno
WHERE e.sal>m.sal;
```

Display the emp records who are working under BLAKE:

EMP_NAME	MGR_NAME

ORACLE STYLE:

```
SELECT e.ename AS emp_name, m.ename AS mgr_name
FROM emp e, emp m
WHERE e.mgr=m.empno AND m.ename='BLAKE';
```

ANSI STYLE:

```
SELECT e.ename AS emp_name, m.ename AS mgr_name
FROM emp e INNER JOIN emp m
ON e.mgr=m.empno
WHERE m.ename='BLAKE';
```

Example:

x.cid < y.cid

GROUPA x

CID	CNAME
10	IND
20	AUS
30	WIN

GROUPA y

CID	CNAME
10	IND
20	AUS
30	WIN

IND VS AUS
IND VS WIN
AUS VS WIN

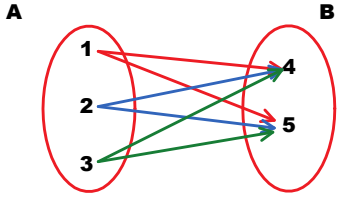
ORACLE STYLE:

```
SELECT x.cname || ' VS ' || y.cname
FROM groupa x, groupa y
WHERE x.cid<y.cid;
```

ANSI STYLE:

```
SELECT x.cname || ' VS ' || y.cname
FROM groupa x INNER JOIN groupa y
ON x.cid<y.cid;
```

A = {1,2,3}
B = {4,5}
AXB = ?



AXB = { (1,4)(1,5)
(2,4)(2,5)
(3,4)(3,5) }

Cross Join / Cartesian Join:

- **In This, Each record in one table will be paired with every record in another table.**
- **In this, don't write Join Condition.**
- **In ANSI STYLE use the keyword: CROSS JOIN**

Example on cross join:

GROUPA a

CID	CNAME
10	IND
20	AUS
30	WIN

GROUPB b

CID	CNAME
40	ENG
50	SL
60	NZ

IND VS ENG
IND VS SL
IND VS NZ
AUS VS ENG
AUS VS SL
AUS VS NZ
WIN VS ENG
WIND VS SL
WIN VS NZ

ORACLE STYLE:

```
SELECT a.cname || ' VS ' || b.cname
FROM groupA a, groupB b;
```

ANSI STYLE:

```
SELECT a.cname || ' VS ' || b.cname
FROM groupA a CROSS JOIN groupB b;
```

JOINS:

- **JOIN is an operation.**
- **one table record will be joined with another table record based on join condition.**

- **GOAL:**
  to retrieve data from multiple tables

Types Of Joins:

Inner Join		matched records only	
	Equi	based on = join operation will be performed	
	Non-Equi	based on other than = join operation will be performed	
Outer Join		matched + unmatched	
	Left outer	matched + unmatched from left	
	Right outer	matched + unmatched from right	
	Full outer	matched + unmatched from left and right	
Self		a table will be joined to itself	
Cross		each record in one table will be joined every record in another table	

Natural Join:
Equi Join without duplicate columns

```
SELECT *
FROM emp e NATURAL JOIN dept d;
```

EMP

EMPNO	ENAME	SAL	DEPTNO
7369	SMITH	800	20
7499	ALLEN	1600	30
7521	WARD	1250	30
7566	JONES	2975	20
7782	CLARK	2450	10
7934	MILLER	1300	10
1001	A	1800	
1002	B	2000	

DEPT

DEPTNO	DNAME	LOC
10	ACCOUNTS	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

Assignment

Thursday, July 25, 2024 8:07 AM

Assignment:

EMPLOYEE e

EMPID	ENAME	PID
1001	A	30
1002	B	30
1003	C	10
1004	D	10
1005	E	
1006	F	

PROJECT p

PID	PNAME
10	X
20	Y
30	Z

Display emp details along with project details => [Equi Join]

empid	ename	pname

Display emp details along with project details
Also display the employees who are on bench [emps who are not participating in any project development]=> [Left Outer Join]

empid	ename	pname

Display emp details along with project details
Also display the projects which are not assigned to any employee => [Right Outer Join]

empid	ename	pname

display the employees who are on bench

empid	ename	pname

Left outer join + condition

display the projects which are not assigned to any employee

empid	ename	pname

Right outer join + condition

display the employees who are on bench.
display the projects which are not assigned to any employee

empid	ename	pname

full outer join + conditions

Display emp details along with project details.
also display the employees who are on bench.
also display the projects which are not assigned to any employee

empid	ename	pname

full outer join

SUB QUERIES

Thursday, July 25, 2024 8:13 AM

Sub Queries:

Syntax:

```
SELECT <columns_list>
FROM <tables_list>
WHERE <column> <operator> (<SELECT QUERY>);
```

Outer query / Main query / Parent query

Inner query / Sub query / Child query

- **A query which is written in another query is called "Sub Query".**
- **Outside query is called "Outer query / main query / parent query".**
- **Inside query is called "Inner query / sub query / child query".**
- **When we don't know filter condition value to find it we write SUB QUERY.**
- **First Inner query gets executed. Then outer query gets executed. The result of inner query will become input for outer query.**
- **Sub query must be written parenthesis.**
- **Sub query must be SELECT query only. It cannot be INSERT / UPDATE / DELETE. Because, Sub query has to find some value. Only SEELCT can find the value.**
- **Outer query can be SELECT / UPDATE / DELETE / INSERT.**
- **We can write 254 sub queries in WHERE clause.**

Types of Sub queries:

- **Non-Correlated Sub Query** **Inner => Outer**
  - **Single Row Sub Query**
  - **Multi Row Sub Query**
  - **Inline View / Inline Sub Query**
  - **Scalar Sub query**
- **Correlated Sub Query** **Outer <=> Inner**

Non-Correlated Sub Query:

- **In Non-Correlated Sub query, first inner query gets executed. Inner query passes value to outer query. Then outer query gets executed.**

- **It has sub types. They are:**
  - **Single Row Sub Query**
  - **Multi Row Sub Query**
  - **Inline View / Inline Sub Query**
  - **Scalar Sub query**

Single row sub query:
if sub query returns 1 row then it is called "Single Row Sub Query"

Examples on Single Row Sub Query:

Display the emp records who are earning more than BLAKE:

BLAKE	**2850**

```
SELECT ename, sal
FROM emp
WHERE sal>(find BLAKE sal);

find BLAKE sal:
SELECT sal FROM emp
WHERE ename='BLAKE';

SELECT ename, sal
FROM emp
WHERE sal>(SELECT sal FROM emp WHERE ename='BLAKE');
```

Display the emp records whose job title is same as BLAKE:

```
SELECT ename, job, sal
FROM emp
WHERE job=(find BLAKE job);

find BLAKE job:
SELECT job FROM emp WHERE ename='BLAKE';

SELECT ename, job, sal
FROM emp
WHERE job=(SELECT job FROM emp WHERE ename='BLAKE');
```

Find 2nd max salary:

```
SAL
----------
4000
3000
5000
6000
2000
```

```
SELECT max(sal)
FROM emp
WHERE sal<6000;
```

```
4000
3000
5000
```

max sal => 5000
2nd max sal

6000
2000

4000
3000
5000
2000

max sal => 5000
2nd max sal

```
SELECT max(sal)
FROM emp
WHERE sal<(find max sal);
```

find max sal:

```
SELECT max(Sal) FROM emp;
```

```
SELECT max(Sal)
FROM emp
WHERE sal<(SELECT max(sal) FROM emp);
```

Find the emp name who is earning max salary:

```
SELECT ename FROM emp
WHERE sal=(find max sal);
```

```
SELECT ename FROM emp
WHERE sal=(SELECT max(sal) FROM emp);
```

Find the emp name who is earning 2nd max salary:

```
SELECT ename FROM emp
WHERE sal=(find 2nd max sal);
```

```
SELECT ename FROM emp
WHERE sal=(SELECT max(sal) FROM emp
WHERE sal<(SELECT max(sal) FROM emp));
```

Find 3rd max salary:

```
SELECT max(sal) FROM emp
WHERE sal<(find 2nd max sal);
```

3000
4000
2500
2000

max sal => 4000
3rd max sal

```
SAL
-------
3000
6000
4000
5000
2500
2000
```

```
SELECT max(sal) FROM emp
WHERE sal<(SELECT max(sal) FROM emp
WHERE sal<(SELECT max(sal) FROM emp));
```

Display most senior record:

```
SELECT * FROM emp
WHERE hiredate=(find most senior's hiredate);
```

```
SELECT * FROM emp
WHERE hiredate=(SELECT min(hiredate) FROM emp);
```

Display junior record in all emps:

```
SELECT * FROM emp
WHERE hiredate=(find most junior's hiredate);
```

```
SELECT * FROM emp
WHERE hiredate=(SELECT max(hiredate) FROM emp);
```

Display seniors of BLAKE:

```
SELECT ename, hiredate FROM emp
WHERE hiredate<(find BLAKE's hiredate);
```

```
SELECT ename, hiredate FROM emp
WHERE hiredate<(SELECT hiredate FROM emp
WHERE ename='BLAKE');
```

Display juniors of BLAKE:

```
SELECT ename, hiredate FROM emp
WHERE hiredate>(find BLAKE's hiredate);
```

```
SELECT ename, hiredate FROM emp
WHERE hiredate>(SELECT hiredate FROM emp
WHERE ename='BLAKE');
```

Find the deptno which is spending max amount on their emps:

```
SELECT deptno FROM emp
GROUP BY deptno
HAVING sum(sal)=(find max amount in all depts sum of salaries);
```

```
SELECT deptno FROM emp
GROUP BY deptno
HAVING sum(Sal)=(SELECT max(sum(Sal)) FROM emp
GROUP BY deptno);
```

Find the dept name which is spending max amount on their emps:

```
SELECT dname FROM dept
WHERE deptno=(find deptno which is spending max amount);
```

```
SELECT dname FROM dept
WHERE deptno=(SELECT deptno FROM emp
GROUP BY deptno
HAVING sum(sal)=(SELECT max(sum(sal)) FROM emp
GROUP BY deptno));
```

Assignment:

- **Find the deptno which is having max no of emps**
- **Find the dept name which is having max no of emps**

Multi Row Sub Query:

- **If sub query returns multiple rows then it is called "Multi Row Sub Query".**

- **For multi row sub query we use following operators:**
  IN
  ALL
  ANY

Examples on Multi Row Sub Query:

Display the emp records whose job title is same as SMITH and BLAKE:

SMITH => CLERK
BLAKE => MANAGER

SELECT ename, job, sal FROM emp
WHERE job IN(find SMITH and BLAKE job titles);

find SMITH and BLAKE job titles:

SELECT job FROM emp
WHERE ename IN('SMITH', 'BLAKE');

SELECT ename,job,sal FROM emp
WHERE job IN(SELECT job FROM emp
WHERE ename IN('SMITH', 'BLAKE'));

Display the emp records who are earning more than SMITH and BLAKE:

SELECT ename, sal
FROM emp
WHERE sal>ALL(find SMITH and BLAKE sal);
sal>ALL(800,2850)

SELECT ename, sal
FROM emp
WHERE sal>ALL(SELECT sal FROM emp
WHERE ename IN('SMITH', 'BLAKE'));

sal>ALL(800,2850) **if sal is > all list of values then condn is TRUE**	**sal>800 AND sal>2850**

then condn is TRUE	

SAL	sal>ALL(800,2850)	

1500	1500	F
3000	3000	T
2500	2500	F
6000	6000	T

Display the emp records whose salary is more than TURNER salary or BLAKE salary:

1500 2850

```
SELECT ename, sal
FROM emp
WHERE sal>ANY(1500,2850);
```

sal>ANY(1500,2850)	sal>1500 OR sal>2850
if sal is > any 1 of the values in list then condn is TRUE	

SAL	sal>ANY(1500,2850)	sal>ALL(1500,2850)

1800	1800 T	1800 F
1000	1000 F	1000 F
3000	3000 T	3000 T
2000	2000 T	2000 F
1200	1200 F	1200 F

```
SELECT ename, sal
FROM emp
WHERE sal>ANY(Find TURNER and BLAKE sals);
```

```
SELECT ename, sal
```

```
FROM emp
WHERE sal>ANY(SELECT sal FROM emp
WHERE ename IN('TURNER', 'BLAKE'));
```

sal IN(2000,3000)	sal=ANY(2000,3000)	sal=2000 OR sal=3000

Inline View / Inline Sub Query:

- If sub query is written in FROM clause then it is called "Inline View".
- This sub query acts like table.
- To control the execution order of clauses we need to write sub query in FROM clause.

Syntax:

```
SELECT <columns_list>
FROM (<Sub Query>)
WHERE <condition>;
```

Execution Order:

FROM
WHERE
GROUP BY
HAVING
SELECT
DISTINCT
ORDER BY
OFFSET
FETCH

Examples on Inline View:

Find 3rd max salary:

```
SELECT ename, sal,
dense_rank() over(order by sal desc) AS rank
FROM emp
WHERE rank=3;
```

Output:

ERROR: RANK invalid identifier

Execution Order:

FROM
WHERE
GROUP BY
HAVING
SELECT
DISTINCT
ORDER BY

Output:
ERROR: RANK invalid identifier

SELECT
DISTINCT
ORDER BY
OFFSET
FETCH

```
SELECT DISTINCT sal
FROM (SELECT ename, sal,
dense_rank() over(order by sal desc) as rank
FROM emp)
WHERE rank=3;
```

Find 5th max sal:

```
SELECT DISTINCT sal
FROM (SELECT ename, sal,
dense_rank() over(order by sal desc) as rank
FROM emp)
WHERE rank=5;
```

Find 10th max sal:

```
SELECT DISTINCT sal
FROM (SELECT ename, sal,
dense_rank() over(order by sal desc) as rank
FROM emp)
WHERE rank=10;
```

Find nth max sal:

```
SELECT DISTINCT sal
FROM (SELECT ename, sal,
dense_rank() over(order by sal desc) as rank
FROM emp)
WHERE rank=&n;
```

Output:
enter .. n: 2
--displays 2nd max sal

```
/
enter .. n:
--displays 5th max sal
```

Find top 3 salaries:

```
SELECT DISTINCT sal
FROM (SELECT ename, sal,
dense_rank() over(order by sal desc) as rank
FROM emp)
WHERE rank<=3;
```

Find top 5 salaries:

```
SELECT DISTINCT sal
FROM (SELECT ename, sal,
dense_rank() over(order by sal desc) as rank
FROM emp)
WHERE rank<=5;
```

Find top n salaries:

```
SELECT DISTINCT sal
FROM (SELECT ename, sal,
dense_rank() over(order by sal desc) as rank
FROM emp)
WHERE rank<=&n;
```

Pseudo Columns: **PSEUDO => FALSE**

ROWNUM
ROWID

ROWNUM => pseudo column

ROW_NUMBER() => analytic function

ROWNUM:

- **It is a pseudo column.**
- **It is used to apply row numbers to records.**
- **Always row numbers will be applied on result of SELECT query.**

Examples on ROWNUM:

Display all emp names and salaries. apply row numbers to them:

```
SELECT rownum, ename, sal
FROM emp;
```

Display the emp names and salaries whose salary is 3000 or more. apply row numbers to them:

```
SELECT rownum, ename, sal
FROM emp
WHERE sal>=3000;
```

Display all columns and rows from emp table. apply row numbers to them:

```
SELECT rownum AS sno, e.* FROM emp e;
```

Display 3rd row from emp table:

```
SELECT *
FROM (SELECT rownum as rn, ename, sal
FROM emp)
WHERE rn=3;
```

*	All columns of sub query

Display 1st row, 5th row and 11th row from emp table:

```
SELECT *
FROM (SELECT rownum as rn, ename, sal
FROM emp)
WHERE rn IN(1,5,11);
```

Display 6th row to 10th row from emp table:

```
SELECT *
FROM (SELECT rownum as rn, ename, sal FROM emp)
WHERE rn BETWEEN 6 AND 10;
```

Display even numbered rows:

```
SELECT *
FROM (SELECT rownum as rn, ename, sal FROM emp)
WHERE MOD(rn,2)=0;
```

Scalar Sub Query:

- If sub query is written in SELECT clause then it is called "Scalar Sub Query".
- This sub query acts like column.

Syntax:

```
SELECT (<sub query>)
FROM <table_name>
WHERE <condition>;
```

Examples on Scalar Sub Query:

Find no of records in emp and dept tables:

```
SELECT (SELECT count(*) FROM emp) AS emp,
(SELECT count(*) FROM dept) AS dept
FROM dual;
```

Output:

```
EMP       DEPT
--------  --------
14        4
```

Find each dept share in salaries:

DEPTNO	SUM_OF_SAL	TOTAL_AMOUNT	PER
10	8750	29025	8750*100/29025 = 30.1464
20	10875	29025	10875*100/29025 = 37.4677
30	9400	29025	9400*100/29025 = 32.3859

```
SELECT deptno, sum(Sal) AS sum_of_sal,
(select sum(sal) from emp) AS total_amount,
TRUNC(sum(sal)*100/(select sum(sal) from emp),2) AS per
FROM emp
GROUP BY deptno
ORDER BY 1;
```

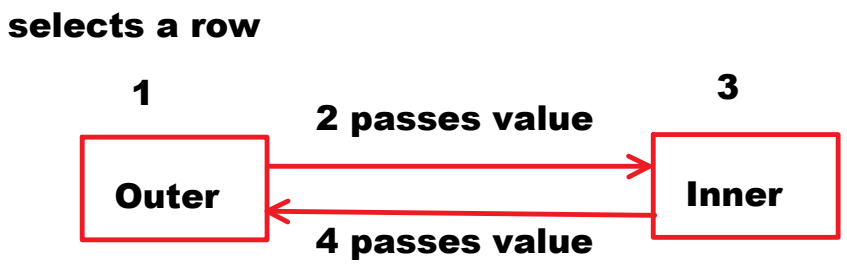
Non-Correlated Sub Query:

- **In Non Correlated sub query, first inner query gets executed. Then outer query gets executed.**
- **In this, Inner query gets executed only 1 time.**

Correlated Sub Query:

- **In Correlated Sub Query, first outer query gets executed. Then Inner query gets executed.**
- **In this, inner query gets executed for multiple times.**

Execution process of Correlated Sub Query:



5 condn => T => selects the row

1. **First, Outer query gets executed. It selects a row.**
2. **Outer query passes value to Inner query.**
3. **Inner query gets executed.**
4. **Inner query passes value to Outer query.**
5. **Outer query condition will be tested. If condition is TRUE, it selects the row.**

These 5 steps will be executed repeatedly for every row selected by outer query

Example:

Display the emp records who are earning more than their dept's avrg

salary:

```
SELECT ename, deptno, sal
FROM emp e
WHERE sal>(SELECT avg(sal) FROM emp WHERE deptno=e.deptno);
```

EMP e

EMPID	ENAME	DEPTNO	SAL
1001	A	10	12000
1002	B	10	8000
1003	C	20	10000
1004	D	20	20000

DEPTNO	AVG_SAL
10	10000
20	15000

ENAME	DEPTNO	SAL
A	10	12000
D	20	20000

Display the emp records who are earning max salary in each dept:

WHERE sal = emp dept's max salary

```
SELECT ename, deptno, sal
FROM emp e
WHERE sal = (SELECT max(sal) FROM emp WHERE deptno=e.deptno);
```

EMP e

EMPID	ENAME	DEPTNO	SAL
1001	A	10	12000
1002	B	10	8000
1003	C	20	10000
1004	D	20	20000

ENAME	DEPTNO	SAL
A	10	12000
D	20	20000

Display the emp records who are senior in each dept:

WHERE hiredate = emp dept's min hiredate

```
SELECT ename, deptno, hiredate
FROM emp e
WHERE hiredate = (SELECT min(hiredate) FROM emp
WHERE deptno=e.deptno);
```

EXISTS:

Syntax:

EXISTS(<Sub Query>)

If sub query selects the rows then it returns TRUE.
If sub query does not select any row then it returns FALSE.

NOT EXISTS:

Syntax:

NOT EXISTS(<Sub Query>)

If sub query does not select any row then it returns TRUE.
If sub query selects the rows then it returns FALSEE.

Display the dept names which are having emps:

```
SELECT dname FROM dept d
WHERE exists(SELECT * FROM emp WHERE deptno=d.deptno);
```

DEPT d

DEPTNO	DNAME
10	ACCOUNTING
20	RESEARCH
30	SALES

EMP e

EMPID	ENAME	DEPTNO	SAL
1001	A	10	12000
1002	B	10	8000
1003	C	20	10000

20	RESEARCH
30	SALES

1002	B	10	8000
1003	C	20	10000
1004	D	20	20000

DNAME
ACCOUNTING
RESEARCH

Display the dept names which are not having emps:

```
SELECT dname FROM dept d
WHERE not exists(SELECT * FROM emp WHERE deptno=d.deptno);
```

Single Row Sub Query:

Set JAMES salary as 30th dept's max salary:

```
UPDATE emp
SET sal=(find 30th dept max sal)
WHERE ename='JAMES';
```

```
UPDATE emp
SET sal=(SELECT max(Sal) FROM emp WHERE deptno=30)
WHERE ename='JAMES';
```

Delete most senior record:

```
DELETE FROM emp
WHERE hiredate=(find most senior's hiredate);
```

```
DELETE FROM emp
WHERE hiredate=(SELECT min(hiredate) FROM emp);
```

CONSTRAINTS

Tuesday, July 30, 2024 7:44 AM

CONSTRAINT:

- **CONSTRAINT => Restrict / Limit / Control**

Max Marks: 100
0 TO 100

CHECK(m1 BETWEEN 0 AND 100)

M1 NUMBER(3)

70
123 ERROR

CHECK(gender IN('M','F'))

GENDER

M
F
Z ERROR

CONSTRAINTS:

- **CONSTRAINT is a rule that is applied on column.**
- **It restricts the user from entering invalid data.**
- **With this, we can maintain accurate and quality data.**
- **Maintaining accurate and quality data is called "Data Integrity".**
- **To implement data integrity feature we use CONSTRAINT.**

ORACLE SQL provides following Constraints:

- **Primary Key**
- **Not Null**
- **Unique**
- **Check**

- **Default**
- **References [Foreign Key]**

Primary Key:

- **It does not accept duplicates.**
- **It does not accept nulls.**
- **When value is mandatory and should not be duplicated then we use PRIMARY KEY.**
- **A table can have only one primary key.**

Example:

Employee

PK

EMPID	ENAME	JOB	SAL	
1234	SAI	CLERK	8000	
1235	KIRAN	CLERK	6000	
1236	SAI	SALESMAN	8000	
	A	MANAGER	12000	ERROR
1234	RAJU	CLERK	7000	ERROR

null

duplicate

Example on Primary Key:

T1

F1	Number(4)	PK

```
CREATE TABLE t1
(
f1 NUMBER(4) PRIMARY KEY
);

INSERT INTO t1 VALUES(1234);
INSERT INTO t1 VALUES(1235);

INSERT INTO t1 VALUES(null);    --ERROR
INSERT INTO t1 VALUES(1234);  --ERROR
```

Example:

T2

F1	NUMBER(4)	PK
F2	VARCHAR2(10)	PK

It is not possible.
A table can have one primary key only

Not Null:

- It does not accept nulls.
- It accepts duplicates.
- When value is mandatory and it can be duplicates then use NOT NULL.

Example:

EMPLOYEE

NOT NULL

EMPID	ENAME	SAL
1234	SAI	7000
1235	SAI	6000
1236		8000

ERROR

null

Example:

```
CREATE TABLE t3
(
f1 NUMBER(4) NOT NULL
);

INSERT INTO t3 VALUES(1);
INSERT INTO t3 VALUES(1);
INSERT INTO t3 VALUES(2);
INSERT INTO t3 VALUES(1);
```

INSERT INTO t3 VALUES(null);
Output:
ERROR

UNIQUE:

- **It does not accept duplicates.**
- **It accepts nulls.**
- **When value is optional and it should not be duplicated then use UNIQUE.**
- **We can insert multiple nulls also.**

Example:

CUSTOMER

UNIQUE

CID	CNAME	MOBILE_NUMBER	
1234	A	9123456789	
1235	B		
1236	C	9123456789	ERROR

UNIQUE

MAIL_ID
----
sai@gmail.com
null
sai@gmail.com ERROR

Example:

```
CREATE TABLE t4
(
f1 NUMBER(4) UNIQUE
);

INSERT INTO t4 VALUES(1);

INSERT INTO t4 VALUES(1);  --ERROR

INSERT INTO t4 VALUES(null);
INSERT INTO t4 VALUES(null);
```

CONSTRAINT	DUPLICATE	NULL
PRIMARY KEY	NO	NO
NOT NULL	YES	NO
UNIQUE	NO	YES

PRIMARY KEY = UNIQUE + NOT NULL

Check:

- **It is used to apply our own condition on column**

Example:

STUDENT

CHECK(m1 BETWEEN 0 AND 100)

SID	SNAME	M1
1234	A	78
1235	B	345 ERROR

ABHI BUS

BUS => 30 seats
1 to 30

CHECK(seat_num between 1 and 30)
SEAT_NUM

25
50

Default:

- **It is used to apply default value to column.**
- **When for almost all records value is same then we set default value for that column.**

Example:

STUDENT default 'NARESH' default 'HYD' default 20000

SID	SNAME	COLLEGE_NAME	CCITY	FEE
1234	A	NARESH	HYD	20000

1235	B	NARESH	HYD	10000

```
INSERT INTO student(sid,sname) VALUES(1234,'A');
INSERT INTO student(sid,sname,fee) VALUES(1235,'B',10000);
```

References [Foreign Key]:

- Foreign Key accepts Primary Key values of another table.
- It accepts duplicates.
- It accepts nulls.
- TO set foreign key we use REFERENCES keyword.

Example:

COURSE

PK

CID	CNAME
10	JAVA
20	C#
30	HTML

STUDENT

FK

REFERENCES COURSE(CID)

SID	SNAME	CID
1001	A	20
1002	B	20
1003	C	10
1004	D	30
1005	E	
1006	F	90 ERROR

Examples on constraints:

Example-1:

USERSINFO

USERID	UNAME	PWD

USERID	don't accept duplicates and nulls	PK

UNAME	don't accept duplicates and nulls	UNIQUE NOT NULL
PWD	password must have min 8 chars	CHECK

```
CREATE TABLE usersinfo
(
userid NUMBER(4) PRIMARY KEY,
uname VARCHAR2(20) UNIQUE NOT NULL,
pwd VARCHAR2(20) CHECK(length(pwd)>=8)
);
```

Example-2:

STUDENT

SID	SNAME	M1

SID	don't accept dups and nulls	PK
SNAME	don't accept nulls	NOT NULL
M1	must be b/w 0 to 100	CHECK

```
CREATE TABLE student
(
sid NUMBER(4) PRIMARY KEY,
sname VARCHAR2(10) NOT NULL,
m1 NUMBER(3) CHECK(m1 BETWEEN 0 AND 100)
);
```

Example-3:

EMPLOYEE

EMPID	ENAME	GENDER	SAL

empid	don't accept dups and nulls	PK
ename	don't accept nulls	NOT NULL
gender	M or F	CHECK
sal	sal must be 5000 or more	CHECK

```
CREATE TABLE employee
```

```
(
empid NUMBER(4) PRIMARY KEY,
ename VARCHAR2(10) NOT NULL,
gender CHAR CHECK(gender IN('M','F')),
sal NUMBER(8,2) CHECK(sal>=5000)
);
```

Example-4:

STUDENT1

SID	SNAME	CNAME	CCITY	FEE

SID	PK
SNAME	NOT NULL
CNAME	DEFAULT 'NARESH'
CCITY	DEFAULT 'HYD'
FEE	DEFAULT 20000

20000.00

```
CREATE TABLE student1
(
sid NUMBER(4) PRIMARY KEY,
sname VARCHAR2(10) NOT NULL,
cname VARCHAR2(10) DEFAULT 'NARESH',
ccity CHAR(3) DEFAULT 'HYD',
fee NUMBER(7,2) DEFAULT 20000
);

INSERT INTO student1 VALUES(1001,'A');
Output:
ERROR: not enough values


INSERT INTO student1(sid,sname) VALUES(1001,'A');
Output:
1 row created
```

Example-5:

DEPT1
PK

DEPTNO	DNAME
10	ACCOUNTS
20	HR
30	SALES

EMP1
PK

FK
REFERENCES dept1(deptno)

EMPNO	ENAME	DEPTNO
1001	A	30
1002	B	10
1003	C	10
1004	D	
1005	E	80 ERROR

```
CREATE TABLE dept1
(
deptno NUMBER(2) PRIMARY KEY,
dname VARCHAR2(10) UNIQUE NOT NULL
);

CREATE TABLE emp1
(
empno NUMBER(4) PRIMARY KEY,
ename VARCHAR2(10) NOT NULL,
deptno NUMBER(2) REFERENCES dept1(deptno)
);
```

NOTE:
PK column data type and FK column data type must be same

Assignment:

Example:

COURSE
PK

CID	CNAME

STUDENT

FK
REFERENCES COURSE(CID)

PK

CID	CNAME
10	JAVA
20	C#
30	HTML

FK
REFERENCES COURSE(CID)

SID	SNAME	CID
1001	A	20
1002	B	20
1003	C	10
1004	D	30
1005	E	
1006	F	90 ERROR

Naming Constraints

Wednesday, July 31, 2024 8:47 AM

Syntax to create Table:

```
CREATE TABLE <table_name>
(
   <column_name> <data_type> [CONSTRAINT <con_name> <con_type>,
   <column_name> <data_type> CONSTRAINT <con_name> <con_type>,
   .
   . ]
);
```

Naming Constraints:

- To identify every constraint uniquely name is required.
- We can give names to constraints.
- If we don't define constraint name implicitly ORACLE defines a constraint name.
- To define constraint name we use CONSTRAINT keyword

Example:

STUDENT5

SID	SNAME	M1

SID	PK	c1
SNAME		
M1	CHECK => must be b/w 0 to 100	c2

```
CREATE TABLE student5
(
sid NUMBER(4) CONSTRAINT c1 PRIMARY KEY,
sname VARCHAR2(10),
```

```
m1 NUMBER(3) CONSTRAINT c2 CHECK(m1 BETWEEN 0 AND 100)
);
```

USER_CONSTRAINTS:

- **it maintains all constraints info**

to see list of constraints of student5 table:

```
SELECT table_name, constraint_name, constraint_type
FROM user_constraints
WHERE table_name='STUDENT5';
```

Constraint can be applied at 2 levels. They are:

- **Column Level Constraint**
- **Table Level Constraint**

Column Level Constraint:

- **If constraint is defined in column definition then it is called "Column Level Constraint".**
- **All 6 constraints can be applied at column level.**

Table Level Constraint:

- **If constraint is defined after defining all columns then it is called "Table Level Constraint".**
- **We can apply only 4 constraints at table level.**
  PK, UNIQUE, CHECK, REFRENCES

Example on Column Level Constraint:

STUDENT6

SID	SNAME	M1

SID	PK	c3
SNAME	NOT NULL	c4
M1	CHECK	c6

```
CREATE TABLE student6
(
sid NUMBER(4) CONSTRAINT c3 PRIMARY KEY,
sname VARCHAR2(10) CONSTRAINT c4 NOT NULL,
m1 NUMBER(3) CONSTRAINT c6 CHECK(m1 BETWEEN 0 AND 100)
);
```

Example on Table Level Constraint:

STUDENT7

SID	SNAME	M1

SID	PK	c7
SNAME	NOT NULL	c8
M1	CHECK	c9

```
CREATE TABLE student7
(
sid NUMBER(4),
sname VARCHAR2(10) CONSTRAINT c8 NOT NULL,
m1 NUMBER(3),
CONSTRAINT c7 PRIMARY KEY(sid),
CONSTRAINT c9 CHECK(m1 BETWEEN 0 AND 100)
);
```

Example:

COURSE5

PK

CID	CNAME
10	JAVA
20	PYTHON
30	C#

STUDENT5

FK

REFERENCES COURSE5(cid)

SID	SNAME	CID	
1001	A	20	
1002	B	20	
1003	C	10	
1004	D	90	ERROR

```
CREATE TABLE course5
(
cid NUMBER(2),
cname VARCHAR2(10),
CONSTRAINT c10 PRIMARY KEY(cid)
);
```

```
CREATE TABLE student5
(
sid NUMBER(4),
sname VARCHAR2(10),
cid NUMBER(2),
CONSTRAINT c11 FOREIGN KEY(cid) REFERENCES course5(cid)
);
```

Why Table Level Constraint?

2 reasons:

- **to apply combination of columns as constraint**
- **to use another column name in constraint**

applying combination of columns as constraint:

Example:

STUDENT
PK(SID, SUBJECT)

SID	SNAME	SUBJECT	MARKS
1001	A	M1	70
1001	A	M2	65
1002	B	M1	55
1002	B	M2	70
1003	C	M1	80
1003	C	M2	45
1001		M1 ERROR	

```
CREATE TABLE student
(
sid NUMBER(4),
```

```
sname VARCHAR2(10),
subject CHAR(2),
marks number(3),
CONSTRAINT c12 PRIMARY KEY(sid, subject)
);
```

Composite Primary Key:
IF we set combination of columns as PK then it is called "Composite Primary Key".

using another column name in constraint:

PRODUCTS

PID	PNAME	MANUFATURED_DATE	EXPIRY_DATE
1001	A	1-AUG-24	25-DEC-23

CHECK(expiry_date>manufactured_date)

```
CREATE TABLE products
(
pid NUMBER(4),
pname VARCHAR2(10),
manufactured_date DATE,
Expiry_date DATE,
CONSTRAINT c13 CHECK(expiry_date>manufactured_date)
);
```

Using ALTER command on Constraints

Thursday, August 1, 2024 8:44 AM

Using ALTER command on Constraints:

USING ALTER command we can:

- **Add the constraints**
- **Rename the constraints**
- **Disable the constraints**
- **Enable the constraints**
- **Drop the constraints**

Syntax of ALTER:

```
ALTER TABLE <table_name> [ADD CONSTRAINT <con_name> <con_type>(<column>)]
                         [RENAME CONSTRAINT <old_name> TO <new_name>]
                         [DISABLE CONSTRAINT <con_name>]
                         [ENABLE CONSTRAINT <con_name>]
                         [DROP CONSTRAINT <con_name>];
```

Example:

STUDENT

SID	SNAME	M1

```
CREATE TABLE student
(
sid NUMBER(4),
sname VARCHAR2(10),
m1 NUMBER(3)
);
```

Add PK to sid:

```
ALTER TABLE student ADD CONSTRAINT c15 PRIMARY KEY(sid);
```

Add NOT NULL to sname:

```
ALTER TABLE student MODIFY sname CONSTRAINT c16 NOT NULL;
```

Note:
Using ADD keyword we can add Table Level Constraints only.
Using MODIFY keyword we can add all Column Level Constraints.

Add check constraint to m1:

```
ALTER TABLE student ADD CONSTRAINT c17 CHECK(m1 BETWEEN
0 AND 100);
```

Rename constraint c15 to z:

```
ALTER TABLE student RENAME CONSTRAINT c15 TO z;
```

Disabling Constraint:

```
ALTER TABLE student DISABLE CONSTRAINT z;
```

Enabling Constraint:

```
ALTER TABLE student ENABLE CONSTRAINT z;
```

Dropping Constraint:

```
ALTER TABLE student DROP CONSTRAINT z;
```

SET OPERATORS

Friday, August 2, 2024 7:46 AM

A = {1,2,3,4,5}
B = {4,5,6,7,8}

A U B = {1,2,3,4,5,6,7,8} = B U A

A UA B = {1,2,3,4,5,4,5,6,7,8} = B UA A

A I B = {4, 5} = B I A

A M B = {1,2,3}
B M A = {6,7,8}

SET OPERATORS:

- **SET OPERATOR is used to combine result of 2 select queries.**

  Syntax:
  <SELECT query>
  <SET OPERATOR>
  <SELECT query>;

- **ORACLE SQL provides following SET OPERATORS:**
  - **UNION**
  - **UNION ALL**

- INTERSECT
- MINUS

UNION:

- It combines result of 2 select queries without duplicates.

UNION ALL:

- It combines result of 2 select queries including duplicates.

INTERSECT:

- It gives common records from the result of 2 select queries

MINUS:

- It gives specific records from first select query result.

Example on SET OPERATORS:

CRICKET

SID	SNAME
1001	A
1002	B
1003	C

FOOTBALL

SID	SNAME
5001	D
1002	B
5002	E

Display all students records who are participating in CRICKET and FOOTBALL:

```
SELECT sid, sname FROM cricket
UNION
SELECT sid, sname FROM football;
```

SID	SNAME
1001	A
1002	B
1003	C
5001	D
5002	E

Display all students records who are participating in CRICKET and FOOTBALL including duplicates:

```
SELECT sid, sname FROM cricket
UNION ALL
SELECT sid, sname FROM football;
```

SID	SNAME
1001	A
1002	B
1003	C

5001	D
1002	B
5002	E

Display the students records who are participating in CRICKET and FOOTBALL:

```
SELECT sid, sname FROM cricket
INTERSECT
SELECT sid, sname FROM football;
```

SID	SNAME
1002	B

Display the students records who are participating in CRICKET only and those students should not be participated in FOOTBALL:

```
SELECT sid, sname FROM cricket
MINUS
SELECT sid, sname FROM football;
```

Output:

SID	SNAME
1001	A
1003	C

Display the students records who are participating in FOOTBALL only and those students should not be participated in CRICKET:

```
SELECT sid, sname FROM football
MINUS
SELECT sid, sname FROM cricket;
```

Output:

SID	SNAME
5001	D
5002	E

Example:

deptno 10	deptno 20
CLERK	CLERK
MANAGER	MANAGER
PRESIDENT	ANALYST
	CLERK
	ANALYST

Display the job titles offered by deptno 10 and 20:

```
SELECT job FROM emp WHERE deptno=10
UNION
SELECT job FROM emp WHERE deptno=20;
```

Display the common job titles offered by deptno 10 and 20:

```
SELECT job FROM emp WHERE deptno=10
INTERSECT
SELECT job FROM emp WHERE deptno=20;
```

Display the job titles offered by deptno 10 and those should not be offered by deptno 20:

```
SELECT job FROM emp WHERE deptno=10
MINUS
SELECT job FROM emp WHERE deptno=20;
```

Display the job titles offered by deptno 20 and those should not be offered by deptno 10:

```
SELECT job FROM emp WHERE deptno=20
MINUS
SELECT job FROM emp WHERE deptno=10;
```

Example:

EMP_US

EMPNO	ENAME
1001	A
1002	B

EMP_IND

EMPNO	ENAME
5001	C
5002	D

Display the emp records who are working from india and us:

```
SELECT * FROM emp_us
UNION
SELECT * FROM emp_ind;
```

Rules of SET OPERATORS:

- **Corresponding columns data types must be same.**

```
SELECT sid, sname FROM cricket
UNION
SELECT sname, sid FROM football;
```

Output:
ERROR

- **Number of columns in both select queries must be same.**

Example:

```
SELECT sid, sname FROM cricket
UNION
SELECT sid FROM fotball;
```

Output:
ERROR

Differences b/w UNION and UNION ALL:

UNION	• it does not give duplicates • slower
UNION ALL	• it gives duplicates • faster

Differences b/w JOINS and SET OPERATORS [UNION]:

JOIN	• it combines the columns • it is used for vertical merging • it can be applied on dissimilar structures
UNION	• it combines the rows • it is used for horizontal merging • it can be applied similar structures

SQL => QUERIES =>
PL/SQL => PROGRAMS =>

ORACLE
DATABASE
TABLES
ROWS & COLUMNS

SQL Commands:

DDL	DRL	DML	TCL	DCL
create alter drop flashback purge truncate rename	select	insert update delete insert all merge	commit rollback savepoint	grant revoke

Built-in functions:

String functions	upper() lower() initcap() Lpad() Rpad() Ltrim() Rtrim() Trim() Substr() Instr()

Conversion	**to_char() to_number() to_date()**
Aggregate / group	**sum() max() min() count() avg()**
Number	**ceil() floor()** **trunc() round() mod()**
Date	**months_between() add_months()** **last_day() next_day()** **sysdate systimestamp**
Analytic	**rank() dense_rank() row_number()**
Special	**NVL() NVL2() user**

Clauses:

FROM
WHERE
GROUP BY
HAVING
SELECT
DISTINCT
ORDER BY
OFFSET
FETCH

Joins:
used to retrieve data from multiple tables

Inner Join => matched records

Equi => based on =

Non-Equi => based on other than =

Outer join => matched + unmatched

Left outer+condn => um from L

Right outer+condn => um from R

Full outer+condns => um from L & R

Self Join

Cross Join

Sub queries:

Non-correlated **inner => outer**

inner => 1time

correlated

CONSTRAINTS

SET OPERATORS