Notes link:
bit.ly/oracledbnotes

oracle software link:
bit.ly/oracle21csoftware

oracle software installation video link:
bit.ly/oracle21cinstallation

Akhil (Admin)
Mobile: 9154156192
(Only Whatsapp)

Oracle SQL & PLSQL @ 11:00 AM IST by Mr Shiva Chaitanya
Day-1 https://youtu.be/_eaGqCtSHWQ
Day-2 https://youtu.be/ImRfKhGUPXY
Day-3 https://youtu.be/OohdYXGEM60
Day-4 https://youtu.be/RxI-Vfb1oh8
Day-5 https://youtu.be/EKwkI1yrRsg
Day-6 https://youtu.be/cnhM7W-k3kQ
Day-7 https://youtu.be/6xxXwfxOVDw
Day-8 https://youtu.be/57-5YwTUMvQ

Steps to attend online session:
https://youtu.be/tyH9zZLPns4

SYLLABUS

Saturday, December 14, 2024 12:14 PM

ORACLE

Module-1 [SQL]: Tables
Module-2: PL/SQL
Module-3 [SQL]: Other DB Objects

Module-1: Tables

SQL Commands	**DDL, DRL, DML, TCL, DCL**
Built-In Functions	**String, Aggregate, Conversion, analytic, Number**
Clauses	**GROUP BY, HAVING, ORDER BY, OFFSET, FETCH, FROM , WHERE**
Joins	**Types of joins: Inner, outer, self, cross**
Sub Queries	**Non-correlated: single row, multi row, inline, scalar correlated**
Set Operators	**Union, Intersect, Minus, Union All**
Constraints	**Primary Key, Foreign Key, Check, Not Null, Unique**

Module-2: PL/SQL

PL/SQL Basics	**Data types, Declare, Assign, print, Read**
Control Structures	**Conditional, Looping, Jumping**
Cursors	**Steps to use cursor, cursor for, inline cursor, Parameterized cursor, ref cursor**
Collections	**Associative array, nested table, v-array**
Exception Handling	**Built-in exceptions, user-defined exceptions**
Stored Procedures	
Stored Functions	
Packages	
Triggers	
Working with LOBs	
Dynamic SQL	

Module-3: Other DB Objects

VIEWS	**Types of Views: Simple, Complex**

INDEXES	**Types of Indexes: B-Tree Index, Bitmap Index**
SEQUENCES	**using sequence, using identity**
MATERIALIZED VIEWS	**Refreshing M.Views**
SYNONYMS	**Making lengthy table names short**

ORACLE

Thursday, December 12, 2024 11:17 AM

Importance of data

Data Store
Database
DBMS
RDBMS
Metadata

BANK

Branches
Customers
Transactions
Employees
Products

Run =>
Opening account
Closing account
Withdraw
Deposit
Fund Transfer

Analyze =>
2022 ?
2023 ?
2024 ?

Amazon

Products
Wishlists
Customers
Suppliers

run

searching for products
Wish list
Placing order

Analyze

2022 ?

2023 ?
2024 ?

GOAL:
Storing org bus data permanently in computer

Different ways of storing data in computer:

- **Variable**
- **Object**
- **File**
- **Database**

Variable:
Variable is temporary.

In java:

int empid=1234;

empid

$\boxed{1234}$

Object:
Object is temporary

e1

Empid	Ename	Job
1234	A	MANAGER

In Java:

```
Class Employee
{
}

Employee e1 = new Employee(1234, ...);
```

File:
File is permanent.

Database:
Database is permanent.

File	• **Developed for 1 user.** • **Small amounts of data.** • **Less security.**
Database [best way]	• **Developed for multiple users.** • **Large amounts of data.** • **More security.**

Data Store:

- **Data Store is a location where data is available.**

  Examples:
  Book, File, Database

Database:

- **Database is a kind of Data Store.**
- **Database is a location where org bus data stored permanently in computer.**
- **Database means, complete information about an organization.**

Example:

BANK DB

Branches
Customers
Transactions
Products

COLLEGE DB

Courses
Students
Fee
Marks

Transactions
Products
.
.

Fee
Marks
.
.
.

DBMS:

- **DBMS => DataBase Management System/ Software.**

- **DBMS is a software that is used to create and maintain the database [org bus data].**

Evolution of DBMSs:

Before 1960s	**Books**
In 1960s	**FMS => File Management Software**
In 1970s	**HDBMS => Hierarchical DBMS** **NDBMS => Network DBMS**
In 1976	**E.F.Codd => RDBMS concept**
Oracle company Founder	**Larry Ellison**
In 1977	**Larry Ellison estd a company** **Software Development Laboratories**
In 1979	**Company name renamed:** **Relational Software Inc.** **Introduced first RDBMS s/w => ORACLE**
In 1983	**Company name renamed:** **ORACLE carp.**

RDBMS:

- **It is a kind of DBMS.**
- **RDBMS => Relational DataBase Management System / Software.**
- **Relation => Table.**
- **RDBMS is a software that is used to create and maintain the database in the form of tables.**

Examples:

ORACLE	**Product of ORACLE company**
SQL SERVER	**MICROSOFT**
DB2	**IBM**
Postgre SQL	**POSTGRE FORUM [a group of companies]**
MY SQL	**SUN MICRO SYSTEMS [ORACLE]**

BANK DB

BRANCHES table

IFSC_CODE	**CITY**	**STATE**	**COUNTRY**

Customers table

Custid	**Cname**	**Mobile**	**Aadhar**	**PAN**	**Mail_id**

Transactions table

Txn_date_time	**Acno**	**Txn_Type**	**Amount**

Employee table

Empid	**Ename**	**Job**	**Sal**

Table:

- **Table is a collection of rows and columns.**
- **Column is vertical representation of data.**
- **Row is horizontal representation of data.**

Example:

Employee $\longrightarrow$ **Table / Relation / Entity**

Example:

Employee → Table / Relation / Entity

EMPID	ENAME	SAL
1001	RAJU	12000
1002	KIRAN	15000

Column / Field / Attribute / Property

Row / Record / Tuple / Entity Instance

Metadata:

- **Metadata is the data about the data.**
- **Example:**
  Table name, Column name, Data type, Field size

EMPLOYEE

EMPID NUMBER(4) -9999 TO 9999	ENAME	SAL
1001	RAJU	12000
RAJU error		
13-DEC-24 error		
6789		
123456 ERROR		

Data Store	• Is a location where data is available. • Example: BOOK, FILE, DB
Database	• Is a kind of Data Store • Is a location => org bus data stored permanently
DBMS	• Is a software • It is used to create and maintain the database

RDBMS	•Is a software •Is a kind of DBMS •It is used to create and maintain the database In the form of tables. Examples: ORACLE, SQL SERVER, DB2
Metadata	•It is the data about the data Example: Table name, column name, data type, field size

ORACLE

Friday, December 13, 2024   11:57 AM

ORACLE:

- **ORACLE is a Relational DataBase Management Software [RDBMS].**

- **It is used to create and maintain the database in the form of tables.**

- **It allows us to store, manipulate and retrieve the data from database.**

  Manipulate => 3 actions => Insert / Update [modify] / Delete

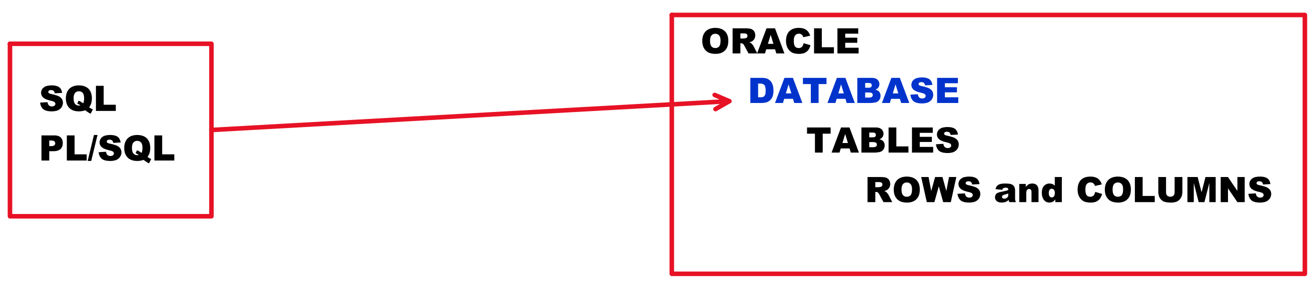
  Emp joined => Insert
  Sal increased => Update
  Emp resigned => Delete

  Retrieve => get back => opening existing data

  Searching for products
  Checking the balance
  Transaction Statement

- **Oracle DB software 2nd version introduced in 1979.**
  First version did not release.

  Latest version: ORACLE 23ai



To communicate with ORACLE,
We can use 2 languages. They are:

- **SQL**
- **PL/SQL**

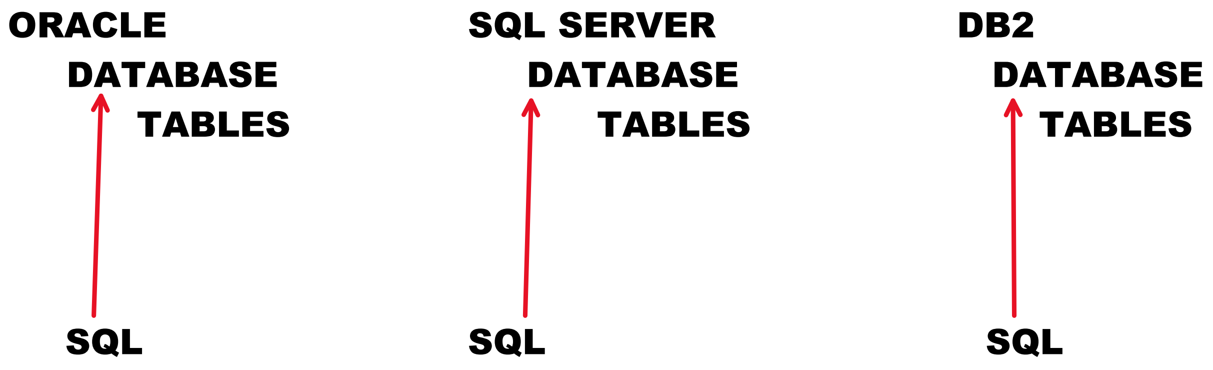
SQL:

- **SQL => Structured Query Language.**
- **It is a Query Language.**
- **In SQL, we develop queries to communicate with ORACLE DB.**

- **Query => request / instruction / command**
- **Query is a request that is sent to DB Server.**
  Example:
  ```
  SELECT ename, sal FROM emp;   --query
  SELECT balance FROM accounts WHERE acno=1234;  --query
  ```

- **SQL is Non-Procedural Language [no programs].**
  In SQL we will not develop any set of statements or programs. Just we develop the queries.

- **SQL is Unified Language.**
  It is common language to communicate with many relational databases.

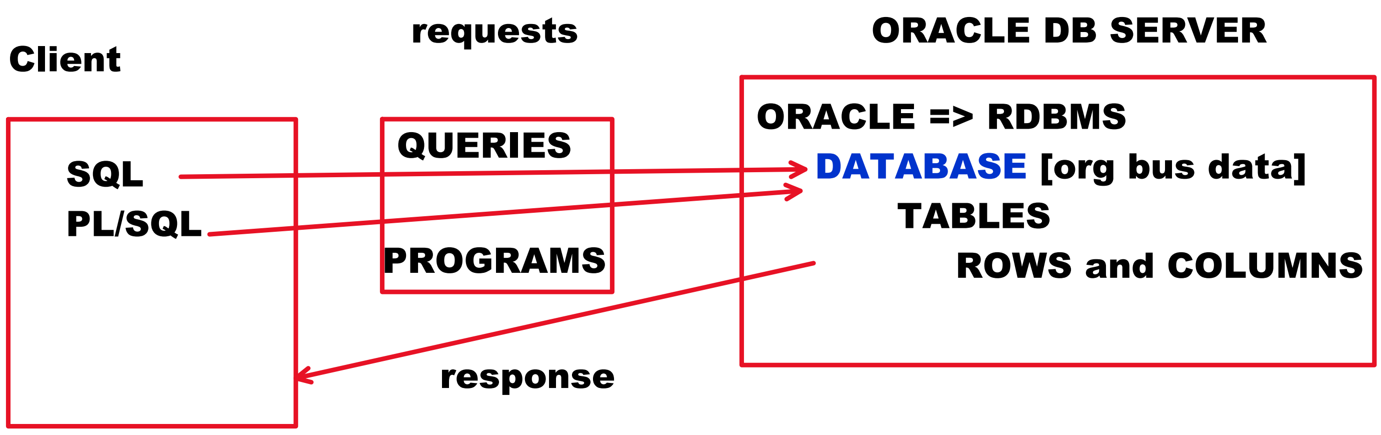


- **To develop the queries SQL provides:**
  - **Commands**
  - **Functions**
  - **Clauses**
  - **Operators**
  - **Constraints**

PL/SQL:

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

- **PL/SQL = SQL + Programming.**
- **PL/SQL is extension of SQL.**
- **PL/SQL is a programming Language.**
- **In this develop the programs to communicate with ORACLE DB.**
- **It is a Procedural Language.**



BANK DB

BRANCHES table

IFSC_CODE	CITY	STATE	COUNTRY

Customers table

Custid	Cname	Mobile	Aadhar	PAN	Mail_id

Transactions table

Txn_date_time	Acno	Txn_Type	Amount

Employee table

Empid	Ename	Job	Sal

SQL Commands

Saturday, December 14, 2024 12:32 PM

SQL Commands:

- **ORACLE SQL provides commands to develop the queries. These commands can be categorized into 5 sub languages. They are:**
  - **DDL**
  - **DRL / DQL**
  - **DML**
  - **TCL**
  - **DCL / ACL**

DDL • **DDL => Data Definition Language** • **Data Definition => metadata** • **It deals with metadata**	**CREATE** **ALTER** **DROP** **FLASHBACK [oracle 10g]** **PURGE [oracle 10g]** **TRUNCATE** **RENAME**
DRL / DQL • **DRL => Data Retrieval Language** • **DQL => Data Query Language** • **It deals with data retrievals.** • **Retrieve => opening existing data**	**SELECT**

DML
 	
• **DML => Data Manipulation Language**
• **Manipulate => 3 actions**
Insert / Update / Delete
	INSERT
	UPDATE
	DELETE
	
	INSERT ALL [oracle 9i]

• **It deals with data manipulations**	**MERGE [oracle 9i]**
TCL • **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**

SQL

Monday, December 16, 2024 11:16 AM

SQL => Query Lang => Queries

ORACLE - SQL Commands:

DDL [metadata]	**DRL [retrievals]**	**DML [manipulations]**	**TCL [transactions]**	**DCL [accessibility]**
Create **Alter** **Drop** **Flashback** **Purge** **Truncate** **Rename**	**Select**	**Insert** **Update** **Delete** **Insert All** **Merge**	**Commit** **Rollback** **Savepoint**	**Grant** **Revoke**

CREATE

Monday, December 16, 2024 11:38 AM

CREATE:

- **It is used to create DB objects like Tables, views, indexes ...etc.**

ORACLE DB objects:

Table
View
Index
Sequence
Synonym
Materialized View

Procedure
Function
Package
Trigger

Table:
Table is a collection of rows and columns.

Example:

Customers

CUSTID	CNAME	CCITY
1234	ABC	HYD
1235	XY	DLH

Syntax to create the table:

```
CREATE TABLE <table_name>
(
```

[ ]	Optional

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

[ ]	Optional
< >	Any

Data Types in ORACLE SQL:

Data Type tells,

- **How much memory has to be allocated**
- **Which type of data should be accepted**

ORACLE SQL provides following data types:

Character Related
 	
• **Used to hold strings**
• **String => is a group of chars**
• **String must be enclosed in single quotes**
 	
Examples:
'RAJU'
'INDIA'
	Char(n)
	Varchar2(n)
	Long
	CLOB
	
	nChar(n)
	nVarchar2(n)
	nCLOB

Integer Related • **Used to hold integers** **Examples:** **1234** **78** **17**	**Number(p)** **Integer** **Int**
Floating point Related • **Used to float values**	**Number(p,s)** **Float** **Binary_Float**

Examples: **12000.00** **2500.80** **67.89**	**Binary_Double**
Date and Time Related • **To hold date and time values** **Examples:** **25-DEC-23** **14-DEC-24 10.30.15.123456 AM**	**Date** **Timestamp (oracle 9i)**
Binary related • **To hold multimedia objects like images, audios, videos ... etc** **Examples:** **Customer img** **Politician speech**	**BFILE** **BLOB**

Character Related Data Types:

Char(n):

- **It is used to hold strings.**
- **It is used to hold fixed length chars.**

Varchar2(n):

- **It is used to hold strings.**
- **It is used to hold variable length chars.**

Examples:

```
COUNTRY_CODE    CHAR(3)
------------------------
IND
AUS
```

```
ENAME   VARCHAR2(15)
-------------
RAJU
NARESH
```

```
VEHICLE_CODE   CHAR(10)
--------------------------
TG09AA1234
TG09AA1235
```

```
MAIL_ID    VARCHAR2(30)
------------------------
sai@gmial.com
Kiran1234@gmail.com
```

```
PAN_NUMBER  CHAR(10)
-----------------------------
ABCDE1234F
```

```
JOB        VARCHAR2(10)
-----------
MANAGER
CLERK
```

Character Related Data Types:

Char(n):

- **n => number of chars**
- **It is used to hold strings.**
- **It is used to hold fixed length chars.**
- **Fixed length data type.**
- **Extra memory will be filled with spaces.**
- **Max memory: 2000 Bytes [2000 chars]**
- **Default size: 1 [CHAR = CHAR(1)]**

Varchar2(n):

- **n => number of chars**
- **It is used to hold strings.**
- **It is used to hold variable length chars.**
- **Variable length data type.**
- **Extra memory will be removed.**
- **Max memory: 4000 Bytes [4000 chars]**
- **Default size: no default size [VARCHAR2 => error]**

NOTE:
VARCHAR2 data type can hold max of 4000 chars only.
To hold more than 4000 chars we can use LONG or CLOB.
CLOB is best to store large amounts of chars.
LONG has some restrictions. To avoid those restrictions CLOB introduced.

LONG:

- **It is used to hold large amounts of chars.**
- **It has some restrictions:**
  - **A table can have only 1 column as LONG type.**
  - **We cannot use built-in functions on LONG type.**

- **Max memory: 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 memory: 4GB**

Normal char set data types: **Char(n) => max 2000 chars** **Varchar2(n) => max 4000 chars** **LONG** **CLOB**	• **ASCII code char data types.** • **Single Byte char data types.** • **These data types can hold english lang chars only.**
National Char set data types: **nChar(n) => max 1000 chars** **nVarchar2(n) => max 2000 chars** **nCLOB** **n => national**	• **UNI code char data types.** • **Multi byte char data types.** • **These data types can hold english + other lang chars also.**

```
In C:     char ch='A';    //1 Byte       => ASCII
In Java:   char ch='A';  //2 Bytes     => UNI
```

ASCII:

- **It is a coding system.**
- **256 chars coded.**

- Code ranges from 0 TO 255.
- ASCII = English + Digits + Special chars.
- 255 => 1111 1111 [8 bits => 1 Byte]

UNI:

- It is a coding system.
- 65536 chars coded.
- Code ranges from 0 TO 65535
- It is extension of ASCII
- UNI = ASCII [eng+dig+special chars] + other language chars
- 65535 => 1111 1111 1111 1111 [16 bits => 2 Bytes]

National Character Set Data Types:

These are used to hold other language chars also.

nChar(n)	• Fixed length data type • n => No of Chars • Max memory: 2000 Bytes [1000 chars]
nVarchar2(n)	• Variable length data type. • Max memory: 4000 Bytes [2000 chars]
nCLOB	• To hold more than 2000 chars we use nCLOB. • Max memory: 4 GB

Integer Related data Types

Wednesday, December 18, 2024 11:28 AM

Integer Related data Types:

NUMBER(p)
Integer
Int

Number(p):

- **P => Precision => Max num of digits**
- **It is used to hold integers.**
- **Max memory: 21 Bytes**
- **P valid range: 1 to 38**

Examples:

```
EMPID   NUMBER(4)          -9999 TO 9999
-------------
1234
1235
123
5
12
7896
9999
10000   ERROR
```

Max marks: 100

```
Maths_Marks  NUMBER(3)    -999 TO 999
-----------------------
78
100
123
999
1000    ERROR
```

```
AGE   NUMBER(2)          -99 TO 99
------
25
99
100     ERROR
```

```
MOBILE_NUMBER   NUMBER(10)

AADHAR_NUMBER NUMBER(12)

CREDIT_CARD_NUMBER NUMBER(16)
```

Note:
Integer = Int = Number(38)

Integer an d Int are alias names of NUMBER(38)

```
CREATE TABLE t15
(
F1 INTEGER,
F2 INT,
F3 NUMBER(38)
);
```

```
DESC T15
Output:
NAME                    TYPE
-------------------------------------------
F1                      NUMBER(38)
F2                      NUMBER(38)
```

```
F3                  NUMBER(38)
```

Floating point related data types

Wednesday, December 18, 2024 12:15 PM

Floating Point related data types:

NUMBER(p,s)
Float
Binary_Float
Binary_Double

NUMBER(p,s):

- **P => Precision => max num of digits**
- **S => Scale => max num of decimal places**
- **It is used to hold float values.**
- **Max memory: 21 Bytes**

Examples:

-999.99 TO 999.99

AVRG NUMBER(5,2)

56.78
567.89
999.99
1000 error
123.5678923 => 123.57
123.5648923 => 123.56

Max marks: 100
5 subjects
500/5 = 100

Max avrg:

$$\boxed{100.\boxed{00}}$$

p=5 **s=2**

-999999.99 TO 999999.99

Max sal:

SAL NUMBER(8,2)

100000.00

999999.99 TO 999999.99 Max sal:

SAL NUMBER(8,2) **100000.00**

```
----------
12000.00
500000.00
1000000.00      ERROR
```

```
-9.9 TO 9.9
HEIGHT   NUMBER(2,1)
----------------
5.2
5.9
5.8
6.0
5.3
9.9
10      ERROR
```

Float	**21 Bytes**
Binary_Float	**4 Bytes**
Binary_Double	**8 bytes**

Date and Time related Data types

Wednesday, December 18, 2024 12:19 PM

Date
Timestamp

Date:

- It is used to hold date values
- Default date format: DD-MON-YY.
- Example: 18-DEC-24
- Date also contains time value.
- It can hold day, month, year, Hours, minutes and seconds.
- It cannot hold fractional seconds.
- Max memory: 7 Bytes.
- Fixed length data type.

18/12/2024 IND date format
DD/MM/YYYY

12/18/2024 US date format
MM/DD/YYYY

ORACLE	DD-MON-YY 18-DEC-24
SQL SERVER	YYYY-MM-DD 2024-12-18

Examples:

```
Date_Of_Birth  DATE
Date_Of_Joining DATE

Ordered_date DATE
Delivery_date DATE
```

Timestamp:

- Introduced in Oracle 9i version.
- It is used to hold date and time values.
- It can hold factional seconds also.
- Fixed length data type.
- Max memory: 11 Bytes

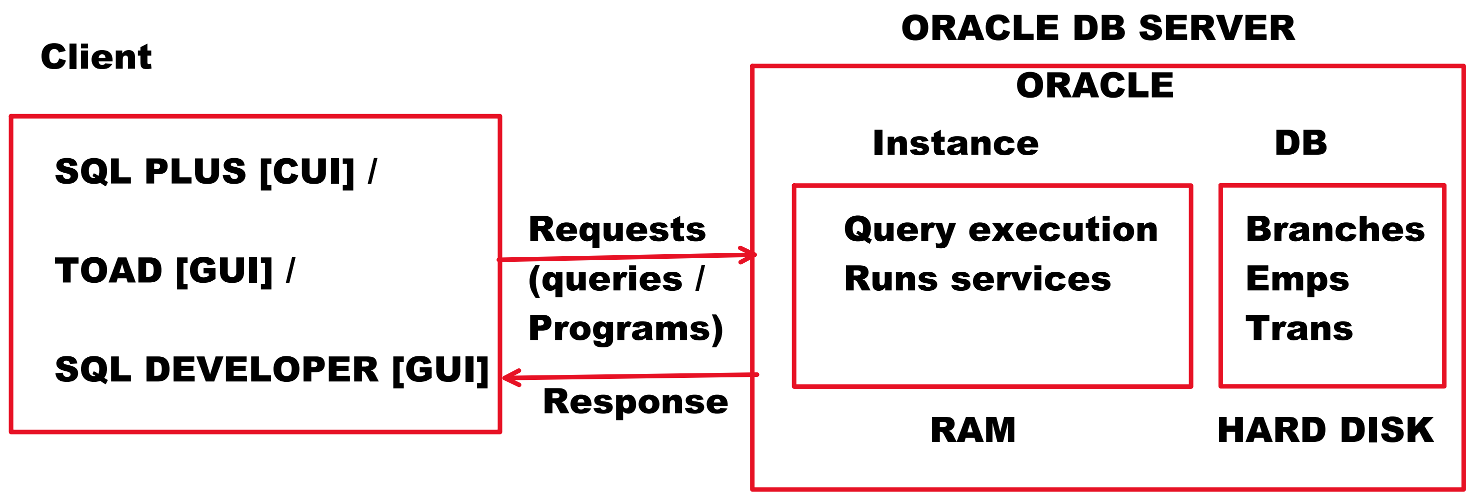
```
TXN_DATE_TIME
------------------------------
18-DEC-24 10.30.15.123456 AM
```

```
ORDERED_DATE_TIME
---------------------------------------
17-DEC-24 2.20.18.567892 PM
```

EMPLOYEE

EMPID NUMBER(6)	ENAME VARCHAR2(20)	PAN CHAR(10)	SAL NUMBER(8,2)	DOJ DATE
123456	RAJU	ABCDE1234F	100000.00	17-AUG-21
123457	SRAVAN			

EMPID NUMBER(6)	LOGIN_DATE_TIME TIMESTAMP
123456	17-DEC-24 10.30.15.123456 AM



NOTE:
When we install ORACLE s/w
Along with ORACLE, SQL PLUS software will be also installed.

ORACLE:

- **ORACLE is server side software.**
- **The machine in which we install ORACLE s/w is called "ORACLE DB SERVER".**
- **DB SERVER contains mainly 2 memories:**
  - **INSTANCE => temporary [RAM]**
  - **DB => permanent [HARD DISK]**

SQL PLUS:

- **SQL PLUS is a client side software.**
- **It is used to connect to ORACLE DB SERVER and communicate With ORACLE DB.**

Opening SQL PLUS:

- **Press Windows+R. It displays RUN dialog box.**
- **Type "sqlplus"**
- **Click on "OK"**

SQL DEVELOPER

Create tables
Views
Indexes
Procedures
Functions
Packages
Triggers

DBA

Installing oracle s/w
Creating user

Security
Backups

Creating User

Thursday, December 19, 2024 12:26 PM

NOTE:
Creating user is duty of DBA [DataBase Administrator].

Syntax to create user:

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

NOTE:

From ORACLE 12c,
There are 2 types of users. They are:
- **Common user   => c##ravi**
- **Local user   => raju**

Example:

Create a user with username c##batch11am,
With the password nareshit:

- **Open sql plus.**

- **Login as DBA:**

  Username: system
  Password: tiger   [At the time oracle installation you have given 1 password]

```
SQL> CREATE USER c##batch11am
        IDENTIFIED BY nareshit;
```

```
            Output:
               User created.

  SQL> GRANT connect, resource, unlimited tablespace
           TO c##batch11am;

           Output:
             Grant succeeded
```

Connect	• **Is a privilege => permission** • **It is a permission for log in**
Resource	• **It is a permission for creating tables, procedures, functions, packages ... etc**
Unlimited tablespace	• **It is a permission for inserting records**

To clear the screen:

[ ]	**Optional**

```
   Syntax:
      CL[EAR] SCR[EEN]

   SQL> CL SCR
             --it clears the creen
```

To see current user name:

```
SQL> SHOW USER
```

Logging in from SQL command prompt:

Syntax:

```
CONN[ECT] <user_name>/<password>
```

Example:

```
SQL> CONN c##batch11am/nareshit
```

Disconnecting from server:

Syntax:

```
DISC[ONNECT]
```

Example:

```
SQL> DISC

SQL> SHOW USER
        --username is empty
```

Modifying Password:

Syntax:

```
ALTER USER <username>
IDENTIFED BY <new_password>;
```

Example:

uname: c##ravi
Pwd: ravi
New password: nareshit

Login as DBA:

Username: system

```
ALTER USER c##ravi
IDENTIFIED BY nareshit;
```

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 <user_name> CASCADE;
```

Example:
Login as DBA:

username: system

```
DROP USER c##ravi CASCADE;
```

Creating Tables and Inserting Records

Friday, December 20, 2024 12:23 PM

Creating Tables and Inserting Records:

CREATE:
CREATE command is used to create the tables.

Syntax:

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

NOTE:
Till ORACLE 21c, a table can have max of 1000 columns only.
In ORACLE 23ai, we can create max of 4096 columns [wide columns].

INSERT:

- **It is used to insert the records into table.**
- **Using INSERT command:**
  - **We can insert single record**
  - **We can insert limited column values**
  - **We can insert multiple records using parameters**

Syntax:

```
INSERT INTO <table_name>[(<column_list>)]
VALUES(<value_list>);
```

Example:

STUDENT

SID	SNAME	AVRG
1234	ABC	67.89
1235	XY	55.66

SID	NUMBER(4)
SNAME	VARCHAR2(10)
AVRG	NUMBER(5,2) max avrg: 100.00

Login as c##batch11am:

```
CREATE TABLE stduent
(
sid NUMBER(4),
sname VARCHAR2(10),
Avrg NUMBER(5,2)
);
Output:
Table created.
```

1234	ABC	67.89
1235	XY	55.66

```
INSERT INTO student
VALUES(1234, 'ABC', 67.89);
Output:
1 row created.

INSERT INTO student
VALUES(1235, 'XY', 55.66);
Output:
1 row created.

COMMIT;

SELECT * FROM student;
```

We can insert limited column values:

1236	RAJU	

```
INSERT INTO student
VALUES(1236, 'RAJU');
```

Output:
ERROR: not enough values

```
INSERT INTO student(sid, sname)
VALUES(1236, 'RAJU');
```

1237		66.55

```
INSERT INTO student(sid, avrg)
VALUES(1237, 66.55);
```

1238		45.23

```
INSERT INTO student(avrg, sid)
VALUES(45.23, 1237);

COMMIT;

SELECT * FROM student;
```

NOTE:
In SQL or PL/SQL, Parameter concept is used to read the value.

Syntax:
&<text>

Examples:
&x
Output:
Enter value for x: [waits to read x]

&sid
Output:
Enter value for sid:

Note:

TO run recent command in memory we use /

/	R[UN]

Inserting multiple records using parameters:

```
SQL> INSERT INTO student VALUES(&sid, '&sname', &avrg);
Output:
Enter value for sid: 5001
Enter value for sname: SAI
Enter value for avrg: 77.66
INSERT INTO student VALUES(&sid, '&sname', &avrg)
INSERT INTO student VALUES(5001, 'SAI', 77.66)

SQL> /
Enter value for sid: 5002
Enter value for sname: RAMESH
Enter value for avrg: 52.12


SQL> /
Enter value for sid: 5003
Enter value for sname: KIRAN
Enter value for avrg: 92.24
```

Example-2:

EMPLOYEE

EMPID	ENAME	STATE	SAL	DOJ
1234	ABC	AP	12000	23-FEB-21
1235	XY	MH	15000	25-DEC-20
1236	AA	TG	13000	Today's date

Empid	NUMBER(4)
Ename	VARCHAR2(10)
STATE	CHAR(2)
SAL 100000.00	NUMBER(8,2)

DOJ	**DATE**

```
CREATE TABLE employee
(
Empid NUMBER(4),
Ename VARCHAR2(10),
State CHAR(2),
Sal NUMBER(8,2),
Doj DATE
);
```

Output:
Table created.

1234	**ABC**	**AP**	**12000**	**23-FEB-21**

```
INSERT INTO employee
VALUES(1234, 'ABC', 'AP', 12000, '23-FEB-2021');
                                    string

DOJ  DATE
--------------------
23-FEB-21     date          Implicit conversion
```

NOTE:
ORACLE supports to implicit conversion.

Don't depend on Implicit Conversion.
It degrades performance.
Always do explicit conversion.

For explicit conversion,
To convert string to date we use to_date() function

1235	**XY**	**MH**	**15000**	**25-DEC-20**

```
INSERT INTO employee
VALUES(1235, 'XY', 'MH', 15000, to_date('25-DEC-2020'));
                                            string
```

DOJ DATE

25-DEC-20 DATE

To_date()
Explicit Conversion

1236	**AA**	**TG**	**13000**	**Today's date**

```
INSERT INTO employee
VALUES(1236, 'AA', 'TG', 13000, sysdate);
```

Sysdate	• It is a built-in function • It returns current system date

Implicit Conversion	If conversion is done implicitly by ORACLE It degrades performance
Explicit Conversion	If conversion is done using function

Example-3:

EMPLOYEE1

EMPID	LOGIN_DATE_TIME
1001	22-DEC-24 10.30.0.0 AM
1002	22-DEC-24 2.20.0.0 PM
1003	Today's date and current time

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

1001	22-DEC-24 10.30.0.0 AM

```
INSERT INTO employee1
VALUES(1001, '22-DEC-2024 10.30.0.0 AM');
```

string

Implicit function

```
LOGIN_DATE_TIME   TIMESTAMP
----------------------------------------------------
22-DEC-2024 10.30.0.0 AM
```

$\xrightarrow{}$ timestamp

1002	22-DEC-24 2.20.0.0 PM

```
INSERT INTO employee1
VALUES(1002, to_timestamp('22-DEC-2024 2.20.0.0 PM'));
```

str

to_timestamp()
Explicit conversion

```
Login_date_time  TIMESTAMP
-------------------------------------------
22-DEC-2024 2.20.0.0 PM
```

$\xrightarrow{}$ timestamp

1003	Today's date and current time

```
INSERT INTO employee1
VALUES(1003, systimestamp);

COMMIT;

SELECT * FROM employee1;
```

Assignment:

Customers

CUSTID	CNAME	AADHAR	MOBILE	PAN	CCITY	CSTATE
1001	ABC	123412341234	9123456789	ABCDE1234F	HYD	AP

Transactions

Custid	T_Date_Time	T_Type	Amount
1001	22-DEC-24 10.30.0.0 AM	W	20000.00
1002	22-DEC-24 2.20.0.0 PM	D	10000.00

Monday, December 23, 2024 11:56 AM

SQL 5 sub languages:

DDL	DRL	DML	TCL	DCL
Create	Select	Insert	Commit	Grant
Alter		Update	Rollback	
		Delete	Savepoint	
Drop				
Flashback		Insert all		
Purge		Merge		
Truncate				
Rename				

To see Table Structure:

Syntax:

```
DESC[RIBE] <table_name>
```

Example:

```
DESC student
Output:
NAME                TYPE
-----------------------------------------
SID                 NUMBER(4)
SNAME               VARCHAR2(10)
AVRG                NUMBER(5,2)
```

1 month ago => student

Desc student

To see tables list which are created by user:

User_Tables:

- **It is a system table / built-in table / readymade table.**
- **It maintains all tables information.**

c##batch11am:

```
DESC user_tables

SELECT table_name
FROM user_tables;
```

Setting Pagesize and Linesize

Tuesday, December 24, 2024 11:33 AM

PAGE

```
1  ABC........................
2  ............................
.
.
.



14
```

SQL> SHOW ALL

PAGESIZE	**14** **[in 1 page => 14 lines]**
LINESIZE	**80** **[in 1 line => 80 chars]**

Setting pagesize:

Syntax:

SET PAGES[IZE] <value>

Example:

SET PAGESIZE 300
(or)
SET PAGES 300

In 1 page, it can display 300 lines

Setting linesize:

Syntax:

SET LINES[IZE] <value>

Example:

```
SET LINESIZE 200
(or)
SET LINES 200
```

In 1 line, it can display 200 chars

```
SET PAGES 300
SET LINES 200
```

(or)

```
SET PAGES 300 LINES 200
```

It is applicable for entire session

```
Login   => session started

SQL> SET PAGES 300  LINES 200

...   session

Logout  => session ended
```

COLUMN ALIAS

Tuesday, December 24, 2024 12:09 PM

COLUMN ALIAS:

- It is used to change column heading in output.
- Alias => another name
- To give column alias we use AS keyword.
- Using AS keyword is optional.

Syntax:

```
<column> [AS] <column_alias>
```

Example:

```
ename AS A
(or(
ename A
```

Display all emp names and salaries.
Display ename column heading as A,
Sal column heading as B

A	B
SMITH	800
ALLEN	1600

```
SELECT ename AS A, sal AS B
FROM emp;
(or)
```

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

DRL

Monday, December 23, 2024 12:03 PM

DRL / DQL:

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

- **It deals with data retrievals.**
- **Retrieve => opening existing data.**

ORACLE - SQL provides only 1 DRL command.
i.e: SELECT

SELECT:

- **It is used to retrieve the data from table.**

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

  Syntax:

```
SELECT <column_list>
FROM <table_name>
[WHERE <condition>];
```

SQL	**ENGLISH**
QUERIES	**SENTENCES**
CLAUSES	**WORDS**

CLAUSE = part of query

Every query is made up with clauses.
Every clause has specific purpose.

All columns and All rows:

Display all columns and all rows of emp table:

```
SELECT *
FROM emp;
```

*	All columns

ORACLE rewrites above query as following:

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

All columns, Specific rows:

Display 7499 emp record:

```
SELECT *
FROM emp
WHERE empno=7499;
```

Specific columns, All Rows:

Display all emp names and salaries:

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

Specific columns, Specific Rows:

Display emp name and salary of empno 7499:

ENAME	SAL

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

SELECT *	**All columns**
SELECT ename, sal	**Specific Columns**
WHERE empno=7499	**Specific row**
Don't write WHERE condition	**All rows**

Operators in ORACLE SQL:

Operator:
Operator is a symbol that is used to perform operations like arithmetic operations 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** **ANY** **ALL** **EXISTS**
Concatenation	**||**
SET OPERATORS	**UNION, UNION ALL, INTERSECT, MINUS**

Arithmetic Operators:

Arithmetic operators are used to perform Arithmetic operations.

In C/Java:

Arithmetic operators are used to perform Arithmetic operations.

+ - * /

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

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

Examples on Arithmetic Operators:

Calculate annual salary of all emps:

ENAME	SAL	SAL*12

```
SELECT ename, sal, sal*12
FROM emp;
```

Calculate annual salary of all emps:

ENAME	SAL	ANNUAL_SAL

```
SELECT ename, sal, Sal*12 AS annual_sal
FROM emp;
```

Calculate annual salary of all emps:

ENAME	SAL	Annual Salary

```
SELECT ename, sal, sal*12 AS Annual Salary
FROM emp;
```

Output:
ERROR

```
SELECT ename, sal, sal*12 AS "Annual Salary"
FROM emp;
```

We enclose column alias in double quotes for 2 purposes:
- **To give alias name in multiple words**
- **To maintain case**

Calculate TA, HRA, TAX and GROSS salaries.
10% on sal => TA
20% on sal => HRA
5% on sal   => TAX
GROSS = sal + Ta + HRA - TAX

ENAME	SAL	TA	HRA	TAX	GROSS

```
SELECT ename, sal,
Sal*0.1 AS TA,
Sal*0.2 AS HRA,
Sal*0.05 AS TAX,
Sal+Sal*0.1+Sal*0.2-Sal*0.05 AS GROSS
FROM emp;
```

Assignment:

STUDENT1

SID	SNAME	M1	M2	M3
1001	A	70	50	80
1002	B	56	88	44

Calculate total marks and avrg marks.

Calculate experience of all emps:

ENAME	HIREDATE	EXPERIENCE

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

Relational Operators / Comparison Operators:

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

- **ORACLE SQL provides following Relational Operators:**
  >   <   >=   <=   =   != / <> / ^=

  Syntax:
  <column> <relational operator> <value>

Examples on Relational Operators:

Display all managers records:

ENAME	JOB	SAL
	MANAGER	

```
SELECT ename, job, sal
FROM emp
WHERE job='manager';
          MANAGER = manager    FALSE
Output:
No rows selected
```

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

Display 7788 emp record:

```
SELECT * FROM emp
WHERE empno=7788;
```

Display BLAKE record:

```
SELECT * FROM emp
WHERE ename='BLAKE';
```

Display 30th dept emp records:

ENAME	SAL	DEPTNO
		30

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

```
DEPTNO              WHERE deptno=30
---------------     -------------------------------
20
30                        20 = 30    F
10                        30 = 30    T
                          10 = 30    F
```

Display the emp records whose salaries are 3000 or more:

ENAME	SAL

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

Display the emp records whose salaries are 1250 or less:

ENAME	SAL

```
SELECT ename, sal
FROM emp
WHERE sal<=1250;
```

Display the emp records who joined after 1981:

```
In 1981                                 CALENDAR order is ASCENDING ORDER

1-JAN-1981                              1-JAN-24      min value
.                                       2-JAN-24
.                                       .
31-DEC-1981                             .
1-JAN-1982                              31-DEC-24     max value
2-JAN-1982      > '31-Dec-1981'         1-JAN-25
.
```

2-JAN-1982 > '31-Dec-1981' 1-JAN-25
.
.

ENAME	SAL	HIREDATE

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

Display the emp records who joined before 1981:

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

ENAME	SAL	HIREDATE

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

NOTE:
String comparison is case sensitive

MANAGER = manager	FALSE
BLAKE = blake	FALSE

CALENDAR ORDER is ASCENDING ORDER.

Display all emp records except managers:

ENAME	JOB	SAL

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

Display all emp records except 30th dept emps:

ENAME	SAL	DEPTNO

```
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 operations.**

- **ORACLE SQL provides following Logical Operators:**
  - **AND**
  - **OR**
  - **NOT**

AND, OR operator are used to separate 2 conditions.

All conditions should be satisfied	AND
At least 1 condition should be satisfied	OR

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:

ENAME	JOB	SAL

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

JOB

MANAGER	T	F	=> FALSE
ANALYST	F	✗	=> FALSE
CLERK	F	✗	=> FALSE

Output:
No rows selected

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

Display the emp records who are working in deptno 10 and 20:

ENAME	SAL	DEPTNO

```
SELECT ename, sal, deptno
FROM emp
WHERE deptno=10 OR deptno=20;
```

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

ENAME	SAL

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

Display the emp records who joined in 1982:

1-JAN-1982
2-JAN-1982
.
.
31-DEC-1982

hiredate>='1-JAN-1982'
AND
Hiredate<='31-DEC-1982'

ENAME	HIREDATE

```
SELECT ename, hiredate
FROM emp
WHERE hiredate>='1-JAN-1982' AND hiredate<='31-DEC-1982';
```

Display 7499, 7698 and 7788 emp records:

```
SELECT *
FROM emp
WHERE empno=7499 OR empno=7698 OR empno=7788;
```

Display the managers who are earning more than 2500:

ENAME	JOB	SAL

```
SELECT ename, job, sal
FROM emp
WHERE job='MANAGER' AND sal>2500;
```

Online shopping

Searching for: dell,   microsoft laptops
Price b/w 50000 to 70000

```
WHERE (cname='DELL' OR cname='MICROSOFT')
AND
(Price>=50000 AND price<=70000)
AND
(color='BLACK' OR color='BLUE');
```

Display the managers records who joined after may 1981:

ENAME	JOB	SAL	HIREDATE

```
SELECT ename, job, sal, hiredate
FROM emp
WHERE job='MANAGER' AND hiredate>'31-MAY-1981';
```

Display the managers whose salary is more than 2500 and those should be joined after april 1981:

ENAME	JOB	SAL	HIREDATE

```
SELECT ename, job, sal, hiredate
FROM emp
WHERE job='MANAGER' AND sal>2500 AND hiredate>'30-APR-1981';
```

Display BLAKE, SCOTT and MILLER records:

```
SELECT * FROM emp
WHERE ename='BLAKE' OR ename='SCOTT' OR ename='MILLER';
```

Display all managers and clerks whose salaries are less than 2500:

ENAME	JOB	SAL

```
SELECT ename, job, sal
FROM emp
WHERE (job='MANAGER' OR job='CLERK') AND sal<2500;
```

ENAME	JOB	SAL
A	MANAGER	2800
B	CLERK	2300
C	MANAGER	2400
D	ANALYST	2000

B	CLERK	2300
C	MANAGER	2400

NOT:

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

NOT truth table:

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

F	NOT(F) => T

Examples on NOT:

Display all emp records except managers:

ENAME	JOB	SAL

```
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
```

Display all emp records except 30th dept emps:

ENAME	SAL	DEPTNO

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

Special Operators / Comparison Operators:

IN:

Display the emp records whose salaries are 1250, 3000 and 5000:

ENAME	SAL

```
SELECT ename, sal
FROM emp
WHERE sal=1250 OR sal=3000 OR sal=5000;
```

(or) **Sal=1250, 3000, 5000**

```
SELECT ename, sal
FROM emp
WHERE sal IN(1250, 3000, 5000);
```

If sal is in list => TRUE
If sal not in list => FALSE

```
SAL                WHERE sal IN(1250, 3000, 5000)
------------        -----------------------------------------------------
3000                       3000    T
4000                       4000    F
5000                       5000    T
1250                       1250    T
1000                       1000    F
```

IN:

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

  Syntax:
  `<column> IN(<value_list>)`

- If column value is in list then condition is TRUE.
- If column value is not in list then condition is FALSE.

  Example:
  `sal IN(1250, 3000, 5000)`

- It avoids of writing multi equality conditions using OR.

Examples on IN operator:

Display all managers and clerks records:

ENAME	JOB	SAL

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

```
JOB
----------
SALESMAN  F
MANAGER    T
ANALYST     F
CLERK         T
```

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

ENAME	SAL	DEPTNo

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

Display the emp records whose empnos are
7499, 7698, 7788:

```
SELECT * FROM emp
WHERE empno IN(7499, 7698, 7788);
```

Display the emp records whose names are BLAKE, SCOTT and MILLER:

```
SELECT * FROM emp
WHERE ename IN('BLAKE', 'SCOTT', 'MILLER');
```

Display all emp records except managers and clerks:

ENAME	JOB	SAL

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

If column value NOT IN list then condn is TRUE
If column value is in list then condition is FALSE

```
JOB                          WHERE job NOT IN('MANAGER' , 'CLERK')
---------                    ---------------------------------------------------------------
```

```
MANAGER                    MANAGER     F
ANALYST                    ANALYST     T
CLERK                      CLERK       F
SALEMSMAN                  SALESMAN    T
```

BETWEEN AND:

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

  Syntax:
  <column> BETWEEN <lower> AND <upper>

  If column value falls under range, condn is TRUE
  If column value does not fall under range, condn is FALSE

Examples on BETWEEN AND:

Display the emp records whose salaries are b/w 2450 and 3000
(or)
Display the emp records whose salaries are 2450 or more and
Those salaries should be 3000 or less:

ENAME	SAL

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

```
SAL            WHERE sal BETWEEN 2450 AND 3000
--------         --------------------------------------------------------
2800                  2800    T
2000                  2000    F
3000                  3000    T
```

Display the emp records who joined in 1982:

ENAME	HIREDATE

1-JAN-1982
2-JAN-1982

ENAME	HIREDATE

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

Display the emp records whose empnos are b/w 7600 to 7800:

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

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

What is the output of above query?

A. Sal b/w 2450 and 3000
B. Error
C. No rows selected
D. All rows selected

Answer: C

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

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

Display the emp records who are not joined in 1981:

```
SELECT ename, hiredate
```

```
FROM emp
WHERE hiredate NOT BETWEEN '1-JAN-1981' AND '31-DEC-1981';
```

LIKE:

In Windows,

To search for all jpg files	***.jpg**
To search for jpg files which are started with 's' letter	**s*.jpg**
To search for jpg files in which 2nd letter is a	**?a*.jpg**

In windows

Wildcard char	Purpose
?	Replaces 1 char
*	Replaces 0 or any

LIKE:

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

  Syntax:
  <column> LIKE <text_pattern>

- **To specify text pattern ORACLE SQL provides following wildcard chars:**

Wildcard char	Purpose
_	Replaces 1 char
%	Replaces 0 or any no of chars

Examples on LIKE:

Display the emp records whose names are started with 'S' letter:

```
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 and ended with 'S':

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

Display the emp records whose name's 2nd letter 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 whose names 3rd letter is A:

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

Display the emp records whose names are having 4 letters:

```
SELECT * FROM emp
```

```
WHERE ename LIKE '____';
```

Display the emp records who joined in DEC month:

ENAME	HIREDATE

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

Display the emp records who are getting 3 digit salary:

ENAME	SAL

```
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%';
```

Display the emp records whose names are not having A letter:

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

Display the emp records whose names are having _:

```
SELECT * FROM emp
WHERE ename LIKE '%\_%' ESCAPE '\';
```

(or)

```
SELECT * FROM emp
WHERE ename LIKE '%$_%' ESCAPE '$';
```

Display the emp records whose names are having %:

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

NULL

Saturday, December 28, 2024 12:20 PM

STUDENT

SID	SNAME	M1 NUMBER(3)
1234	A	70
1235	B	0
1236	C	55
1237	D	

Unable to insert ABSENT
So, insert NULL

EMPLOYEE

EMPID	ENAME	SAL
1001	A	15000
1002	B	20000
1003	C	

NULL
Sal value unknown

NULL:

- **NULL means empty / blank / no value.**
- **When we are unable to insert the value or when value is unknown we insert NULL.**
- **NULL is not equals to 0 or space.**
- **If NULL is participated in arithmetic operation then result will be NULL.**

  Example:

SELECT 100+200 FROM dual;
Output:
300

SELECT 100+200+null FROM dual;
Output:
Null

NOTE:
DUAL is a readymade table.
It has 1 column and 1 row.
To work with non-table data we use DUAL.
Till ORACLE 21c, FROM is mandatory

DUAL

DUMMY
X

In ORACLE 23ai,
FROM clause made as optional.

SELECT 100+200;
Output:
300

Display the emp records who are
getting comm as 500:

```
SELECT ename, sal, comm
FROM emp
WHERE comm=500;
```

Display the emp records who are

getting comm as null:

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

Null = Null	**FALSE**
Null != Null	**FALSE**

- **For NULL comparison we use IS NULL**

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

```
SELECT ename, sal, comm
FROM emp
WHERE comm IS null;
Output:
Displays wose comm is null
```

How to insert null?