Online students:
ADMIN KHUSHI (7997993914)
Whatsapp only

Offline students:
ADMIN MANOJ
9154861171 (whatsapp only)

Temporary notes link: [first 10 days only]
bit.ly/oracledbnotes

oracle software link:
bit.ly/oracle21csoftware

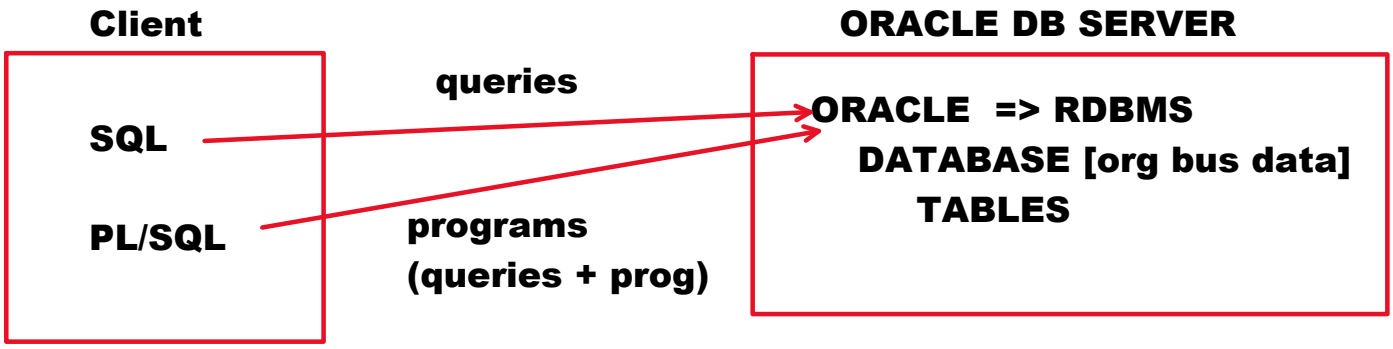
oracle software installation video link:
bit.ly/oracle21cinstallation

Day-1:- https://youtu.be/Aw2GX5n2Ryo
Day-2:- https://youtu.be/qtbvsIVjEM8
Day-3:- https://youtu.be/CiJ8LtjIsas
Day-4:- https://youtu.be/cK53M7VsIfE
Day-5:- https://youtu.be/7h1AtLLNdkI

SYLLABUS

Thursday, July 3, 2025 10:02 AM

ORACLE



ORACLE

Module-1: TABLES [SQL]
Module-2: PL/SQL
Module-3: OTHER DB OBJECTS [SQL]

Module-1: TABLES [SQL]

SQL commands	DDL, DRL, DML, TCL, DCL
Built-In Functions	String functions, Conversion , Date, Number, Analytic, aggregate, ...
Clauses	Group By, Order By, Having, Offset, fetch, ...
Joins	Inner Join => equi, non-equi Outer => left, right, full Self Cross
Sub Queries	Non-correlated single row, multi row, scalar, inline view Correlated

Constraints	**Primary key, check, Foreign key, ...**
Set Operators	**Union, Union All, Intersect, Minus**

Module-2: PL/SQL

PL/SQL Basics	**Data types,** **Declare, assign, print, read** **Using SQL commands in PL/SQL**
Control Structures	**Conditional** **Looping** **Jumping**
CURSORS	**Steps, cursor attributes,** **Types of cursors** **Simple cursor, ref cursor** **Cursor for loop** **Inline cursor** **Parameterized cursor**
COLLECTIONS	**Associative array, nested table, v-array**
EXCEPTION HANDLING	**Built-in exceptions** **User-defined exceptions** **Raise** **Raise_Application_error()** **Pragma exception_init()**
STORED PRCOEDURES	
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** **Simple, composite, function-based, unique** **Bitmap index**
Sequences	**sequence, identity, ..**
M.views	**Refresh ways**
Synonyms	

SQL DEVELOPER (or) PL/SQL DEVELOPER

ORACLE

Wednesday, July 2, 2025 9:20 AM

ORACLE

DB SERVER

SQL
PL/SQL

ORACLE
DATABASE
TABLES

Importance of data

Data Store
Database
DBMS
RDBMS
Metadata

Importance of data:

BANK

Branches
Customers
Transactions
Products
Creditcards
.
.

Opening account
Deposit
Withdraw
Fund Transfer
Closing Account

RUN

2022 ?
2023 ?
2024 ?
2025 ?

ANALYZE

GOAL:

Storing business data permanently in computer

VARIABLE	**It is temporary**
FILE	**It is permanent**
DATABASE	**It is permanent**

DATABASE is best one to store business data permanently in computer.

FILE	DATABASE
• We can store small amounts of data • Less security • Less performance	• We can store large amounts of data • More security • More performance

Data Store:

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

Examples:

BOOKS, FILES, DATABASES

DATABASE:

- **DATABASE is a kind of data store.**
- **It is a location where organization's business data stored permanently in computer.**

DATABASE => org bus data

BANK DB

BRANCHES

IFSC_CODE	CITY	STATE

CUSTOMERS

CUSTID	CNAME	MOBILE	PAN	..	..

TRANSACTIONS

TXN_ID	T_DATE_TIME	T_TYPE	ACNO	AMOUNT	..

EMPS

EMPID	ENAME	JOB	SAL	..

- **DATABASE contains interrelated data in an organized form [table].**
- **organized form => meaningful form.**

BANK DB	COLLEGE DB
Bank related data	College related data
Branch	COURSE
Customer	STUDENT
Transaction	MARKS
	FEE

DBMS:

- **DBMS => DATABASE MANAGEMENT SYSTEM/SOFTWARE**
- **It is a software.**
- **It is used to create and maintain the database.**

Evolution of DBMS:

Before 1960s	BOOKS
In 1960s	FMS [File Management s/w]
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	LARRY ELLISON Renamed company: Relational Software Inc. Released ORACLE s/w
In 1983	LARRY ELLISON Renamed company: ORACLE corp.

RDBMS:

- It is a kind of DBMS.
- It is a software.
- It is used to create and maintain the database in the form of tables.

BRANCHES table

IFSC_CODE	CITY	STATE
SBI000001234	HYD	TG

CUSTOMERS table

CUSTID	CNAME	..	..
12345678	KIRAN		

.
.

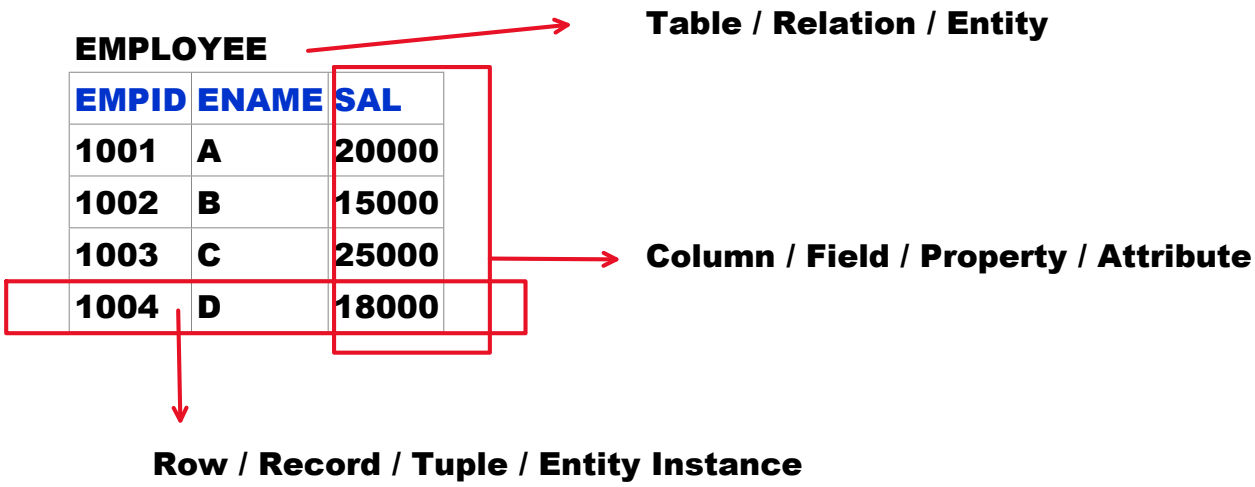
Examples:
ORACLE, DB2, Postgre SQL, SQL SERVER, MY SQL

ORACLE	Product of ORACLE company
DB2	IBM

Postgre SQL	**Postgre Forum**
SQL SERVER	**MICROSOFT**
MY SQL	**SUN MICRO SYSTEMS [ORACLE]**

Data Store	LOCATION	Data is available	BOOKS, FILES, DATABASES
Database	Data Store	Loc => org bus data stored permanently	
DBMS	Software	To maintain the database	
RDBMS	Software	To maintain the database in the form of tables	ORACLE, SQL SERVER, DB2, ..

Table:



- Table is a collection of rows and columns.
- Column => vertical representation of data.
- Row => Horizontal representation of data.

Metadata / Data Definition:

- Metadata is the data about the data.
- It tells about the data.

- **Examples:**
  Column name, Table name, Data type, field size, ...etc

EMPLOYEE

EMPID NUMBER(4) -9999 TO 9999	ENAME	SALARY
1001	A	20000
1002	B	15000
RAJU error		
25-DEC-2024 error		
123		
12345 ERROR		
9999		
10000 ERROR		

ORACLE

Thursday, July 3, 2025 10:32 AM

ORACLE:

- **ORACLE is a Relational DataBase Management Software [RDBMS].**
- **It is used to create and maintain the database in the form of tables.**

- **Using ORACLE DB s/w we can store, manipulate and retrieve the data of database.**

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

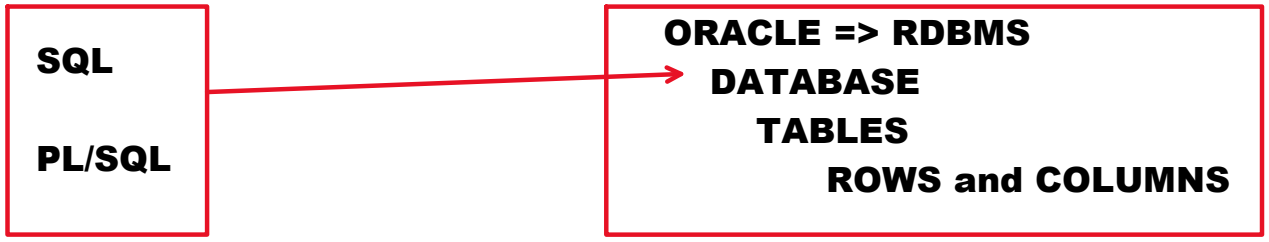
  Emp joined => INSERT
  Sal increased => UPDATE
  Emp resigned => DELETE

  Retrieve => Getting back => opening existing data

  Checking balance
  Transaction statement
  Searching for products

- **ORACLE software 2nd version released in 1979.**
  They didn't release 1st version.

- **Latest version is: ORACLE 23ai**



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

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

SQL:

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

SQL => QUERIES => ORACLE
(requests)

- **Query => request / command / instruction**

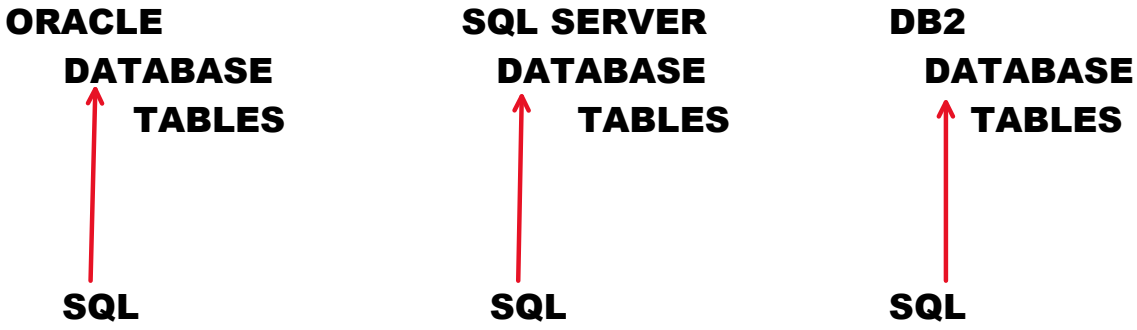
- **Query is a request that is sent DB SERVER.**

Examples:

```
SELECT balance
FROM accounts
WHERE acno=1234;

SELECT ename, sal
FROM emp;

SELECT ename, sal
FROM emp
WHERE sal>12000;
```

- **SQL is non-procedural language.**
  In SQL, we will not develop any set of statements or programs. Just we develop queries.

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



- **SQL provides commands.**
- **SQL provides functions.**
- **.**
- **.**

PL/SQL:

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

- **It is a programming language.**
- **In this, we develop the programs to communicate with ORACLE.**

PL/SQL => PROGRAMS => ORACLE

- **PL/SQL = SQL + Programming**
- **PL/SQL is extension of SQL.**

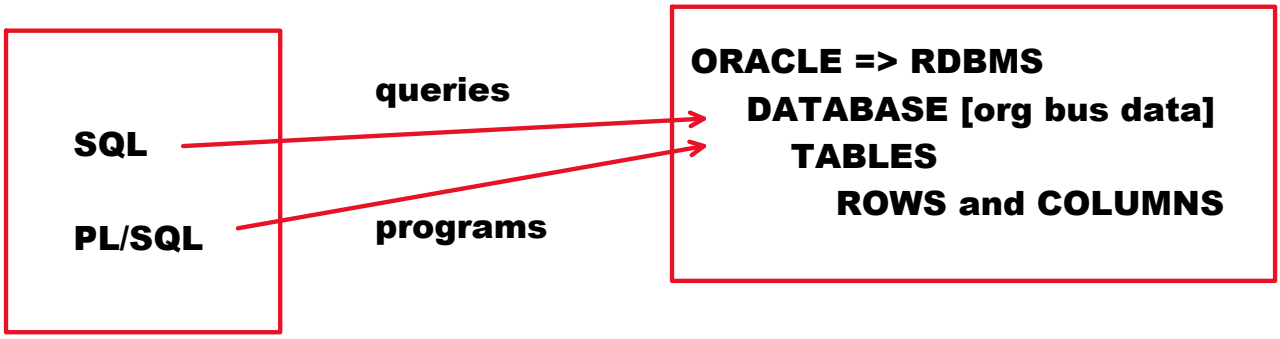
PL/SQL program

Queries => SQL statements
+
Programming => PL/SQL statements

- **PL/SQL program improves the performance.**

SQL COMMANDS

Monday, July 7, 2025 9:25 AM



SQL commands:

SQL => QUERIES => ORACLE

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

DDL • **DDL => Data Definition Language** • **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 => getting data back** **Opening existing data**	**Select**
DML • **DML => Data Manipulation Language** • **Manipulate => 3 actions** **[INSERT / UPDATE / DELETE]**	**Insert** **Update** **Delete** **Insert All [Oracle 9i]** **Merge [Oracle 9i]**
TCL • **TCL => Transaction Control Language** • **Transaction => series of actions [SQL commands]**	**COMMIT** **ROLLBACK** **SAVEPOINT**
DCL/ACL • **DCL => Data Control Language** • **ACL => Accessing Control Language** • **It deals with data accessibility.**	**GRANT** **REVOKE**

SQL commands:

DDL metadata	DRL retrievals	DML manipulations	TCL Transactions	DCL

Create **Alter** **Drop** **Flashback** **Purge** **Truncate** **Rename**	**Select**	**Insert** **Update** **Delete** **Insert All** **Merge**	**Commit** **Rollback** **Savepoint**	**Grant** **Revoke**

CREATE

Monday, July 7, 2025 10:12 AM

CREATE:

- **CREATE command is used to create the Database Objects like tables, views, indexes ..etc.**

Database Objects:

TABLES
VIEWS
INDEXES
MATERIALIZED VIEWS
SEQUENCES
SYNONYMS
PROCEDURES
FUNCTIONS
PACKAGES
TRIGGERS
.
.

Syntax to create the Table:

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

[ ]	Optional
< >	Any

EMPLOYEE

EMPID	ENAME	SAL

ORACLE 21c	**Till oracle 21c,** **A table can have max of 1000 columns**
ORACLE 23ai	**In Oracle 23ai,** **A table can have max of 4096 columns**

Data Types in ORACLE SQL:

Data type mainly tells,

- **How much memory should be allocated**
- **Which data should be accepted**

ORACLE SQL provides following data types:

Character Related • **Used to hold strings.** • **String => a group of chars** • **Can accept letters, digits and special chars** **Examples:** **'RAJU'** **'INDIA'** **'TG'**	**Char(n)** **Varchar2(n)** **Long** **CLOB** **nChar(n)** **nVarchar2(n)** **nCLOB**
Integer Related • **Used to hold integers.** **Examples:** **5001** **19** **892** **78**	**Number(p)** **Integer** **Int**

Floating point Related • Used to hold float values **Examples:** 20000.00 2500.80 56.78	**Number(p, s)** **Float** **Binary_Float** **Binary_Double**
Date and Time Related • Used to hold date or datetime values **25-DEC-23** **7-JUL-25 10.49.15.123456 AM**	**Date** **Timestamp**
Binary Related • Used to hold multimedia objects Like images, audios, videos, documents, ... etc	**BFILE** **BLOB**

Character related data types:

Char(n):

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

Varchar2(n):

- n => number of chars.
- It is variable length data type.

- **It is used to hold variable length chars.**
- **Extra memory will not be filled with spaces.**
- **Max memory: 4000 bytes [4000 chars]**
- **Default size: no default size**

Examples:

State_code CHAR(2)
TG
AP
UP

ENAME VARHCHAR2(10)
SAI
NARESH
KIRAN

PAN_NUMBER CHAR(10)
ABCDE1234F
QWERT3456Z

PNAME VARCHAR2(20)
KEYBOARD
MOUSE

GENDER CHAR
M
F

MAIL_ID VARCHAR2(30)
sai_kumar123@gmail.com
raj@yahoo.com

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 one.

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: 2 GB

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 function on CLOB type.
- Max memory: 4 GB

Character Related data types:

Char(n) Varchar2(n) Long CLOB	• ASCII code char data types. • Single Byte char data types. • These can hold ENGLISH lang chars only.
nChar(n) nVarchar2(n) nCLOB n => national char set data types	• UNI code char data types. • Multi Byte char data types. • These can hold other language chars also.

In C:

```
char  ch='A';   // 1 Byte
```

In java:

```
char ch='A';   // 2 Bytes
```

ch

A

ASCII:

- **It is a coding system.**
- **256 chars coded.**
- **Ranges from 0 TO 255.**
- **255 => 1111 1111 [8 bits => 1 Byte]**
- **ASCII = english letters + digits + special chars**

UNI:

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

nChar(n)	**•Fixed length data type** **•Max memory: 2000 Bytes [1000 chars]**
nVarchar2(n)	**•Variable length data type.** **•Max memory: 4000 Bytes [2000 chars]**
nCLOB	**•Variable length data type.** **•Max memory: 4 GB** **•To hold more than 2000 chars we use it.**

Integer related data types:

Number(p):

- **p => precision => max number of digits**
- **Used to hold integers.**
- **p valid range: 1 TO 38.**
- **Max memory: 22 Bytes.**

Examples:

```
EMPID   NUMBER(4)          -9999 TO 9999
----------
1234
1235
123
12
1
9999
10000     ERROR
123456    ERROR
```

```
CUSTID     NUMBER(6)
---------------
123456
123457
```

```
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)

T41

F1 INTEGER	F2 INT	F3 NUMBER(38)

```
CREATE TABLE t42
(
 f1 INTEGER,
 f2 INT,
 f3 NUMBER(38)
);
```

Output:

Table created.

```
DESC  t42
```

Output:

NAME	TYPE
F1	NUMBER(38)
F2	NUMBER(38)
F3	NUMBER(38)

Floating point related data types:

Number(p,s):

- **p => precision => max number of digits [including dec places]**
- **s => scale => max number of decimal places**
- **It is used to hold float values.**

Examples:

-999.99 TO 999.99

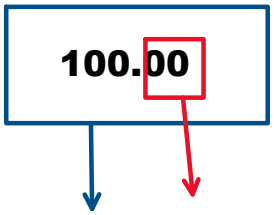
AVRG NUMBER(5,2)

56.78
123.67
999.99
1000 ERROR
123.67893452 => 123.68
786.56345789 => 786.56

654.78923456 => 654.79
654.78423456 => 654.78

Max marks: 100
5 subjects => 500
500/5 = 100

MAX AVRG:



s=2

p=5

-999999.99 TO 999999.99

SAL NUMBER(8,2)

40000.00
600000.00
1000000.00 ERROR

Max sal:

100000.00

-9.9 TO 9.9

HEIGHT NUMBER(2,1)

5.5
8.5
10.0 ERROR

Float	22 Bytes
Binary_Float	4 Bytes
Binary_Double	8 Bytes

Date and Time Related data types:

Date:

- It is used to hold date values.
- Default oracle date format: DD-MON-YY.
- Example: 10-JUL-25
- It can hold day, month, year, hours, minutes and seconds.
- It cannot hold fractional seconds.
- Memory: 7 Bytes.
- It is Fixed length data type.

Timestamp:

- It is used to hold date and time.
- Default format: DD-MON-YY HH.MI.SS.FFFFFF AM
- Example: 10-JUL-25 09.30.15.123456 AM
- It can hold day, month, year, hours, minutes, seconds and fractional seconds.
- It is extension of DATE type.
- Memory: 11 Bytes.
- It is Fixed length data type.

Examples:

```
Txn_date  DATE
----------
10-JUL-25
11-JUL-25
```

ORDERED_DATE_TIME TIMESTAMP

10-JUL-25 10.30.15.123456 AM
11-JUL-25 02.20.00.000000 PM

EMPLOYEE

EMPID NUMBER(6)

123456
123457

ENAME VARCHAR2(15)

KIRAN
NARESH

ECITY CHAR(3)

HYD
BLR

EXP_SUMMARY CLOB

Gfdh hfddhdsh

SAL NUMBER(8,2)

100000.00

Date_Of_Joing DATE

25-DEC-23
17-FEB-20

LOGIN_DATE_TIME TIMESTAMP

10-JUL-25 10.30.15.123456 AM
11-JUL-25 02.20.45.345678 PM

T1

	F1 Char(10)	F2 Varchar2(10)	
10	RAJU6spaces	RAJU	4
10	SAI7spaces	SAI	3

Fixed length Variable length

To find string length we use length() function

Examples:

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

```
SELECT f1, length(f1), f2, length(f2)
FROM t1;
```

Output:

F1	LENGTH(f1)	F2	Length(f2)
RAJU6spaces	10	RAJU	4
SAI7spaces	10	SAI	3

```
SELECT f1 || 'hello', f2 || 'hello' FROM t1;
```

Output:

f1 || 'hello'	f2 || 'hello'
RAJU hello	RAJUhello
SAI hello	SAIhello

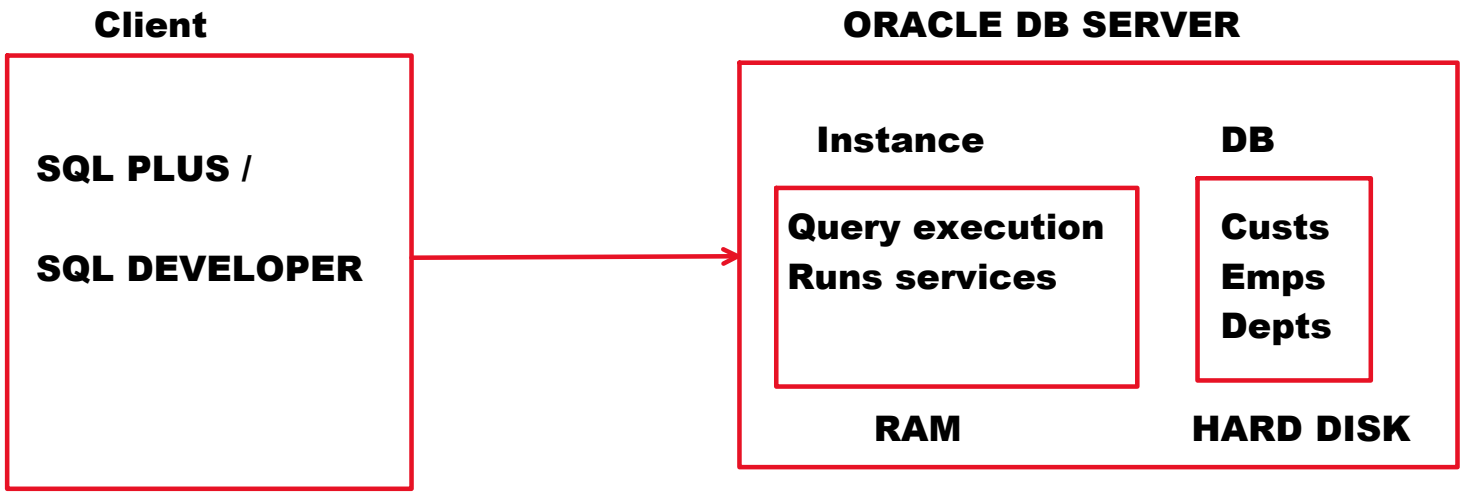
SQL DEVELOPER
(or)
PL/SQL DEVELOPER

Creating tables
Creating views
Creating Indexes
Creating Materialized views
Create procedures
Create functions
Create triggers
.
.

DBA
[database administrator]

Installing oracle s/w
Creating users
Security
Backups and Recovery
.
.
.

Client-Server Architecture:



DB SERVER = INSTANCE + DB

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

ORACLE:

- **It is server side software.**
- **The machine in which we install ORACLE s/w is called "ORACLE DB SERVER".**
- **DBA installs ORACLE s/w in SERVER.**

- **DB SERVER = INSTANCE + DB.**

- **DB SERVER contains 2 memories. They are:**

- **INSTANCE [RAM]**
- **DATABASE [HARD DISK]**

SQL PLUS:

- **It is client side software.**
- **It is used to connect to DB SERVER and communicate with ORACLE DB.**

Opening SQL PLUS:

- **Press windows+R. It displays RUN dialog box.**
- **Type "sqlplus".**
- **Click on "OK". It displays SQL PLUS window.**

Login as DBA:

Enter user name: system
Enter password: tiger [At the time of ORACLE installation you have given password. Enter that password here]

SQL> -- type the queries

Creating user

Saturday, July 12, 2025 9:42 AM

Creating user:

Syntax:

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

Example:

Create a user with the username "c##naresh"
With the password "naresh":

- **Open SQL PLUS.**
- **Login as DBA.**

Username: system
Password: tiger [at the time of ORACLE installation you have given password. Enter that password].

```
SQL> CREATE USER c##naresh
        IDENTIFIED BY naresh;
```

Output:
User created.

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

Output:
Grant succeeded.

To login from SQL command prompt:

Syntax:

CONN[ECT] <username>/<password>

Example:

SQL> CONN c##naresh/naresh

Output:

Connected.

To see current user name:

SQL> SHOW USER

TO disconnect from SERVER:

Syntax:

DISC[ONNECT]

SQL> DISC

Output:

Disconnected from ORACL DB

To clear the screen:

Syntax:

CL[EAR] SCR[EEN]

Example:

SQL> CL SCR

Modifying user's password:

Syntax:

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

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

Example:

Login as DBA:

```
ALTER USER c##naresh
IDENTIFIED BY tiger;
Output:
User Altered.
```

Modifying DBA's password:

Username: sys as sysdba
Password: [don't enter any password.press enter key]

```
SQL> ALTER USER system
        IDENTIFIED BY nareshit;
Output:
User Altered.
```

Dropping User:

Syntax:

```
DROP USER <username> CASCADE;
```

Example:

Login as DBA:

SQL> DROP USER c##naresh CASCADE;
Output:
User dropped.

All_users:

- **It is a system table / built-in table / readymade table.**

  SQL> DESC all_users

  SQL> SELECT username FROM all_users;

SQL => Queries => ORACLE => RDBMS
DATABASE
TABLES
rows and cols

DDL	DRL	DML	TCL	DCL
Create Alter Drop Flashback Purge Truncate rename	Select	Insert Update Delete Insert all Merger	Commit Rollback savepoint	Grant Revoke

CREATE

Monday, July 14, 2025 10:03 AM

CREATE:

- **CREATE command is used to create the tables.**

Syntax:

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

< >	Any
[ ]	Optional

STUDENT

SID	SNAME	AVRG
1001	ABC	67.89
1002	XY	55.66

INSERT:

- **INSERT command is used to insert the records [rows].**
- **Using INSERT command we can insert:**
  - **Single record**
  - **Limited column values**
  - **Multiple records using parameters**

Syntax:

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

Example-1:

STUDENT

SID	SNAME	AVRG

1001	ABC	67.89
1002	XY	55.66

- Open SQL PLUS.
- Login as c##batch9am.

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

Output:

Table created.

=> Table created permanently in DB.
=> CREATE = CREATE + COMMIT.
=> CREATE is auto commit command.
=> Every DDL command is auto commit command.

NOTE:
SQL is not case sensitive language.
CREATE = create = CReaTe

1001	ABC	67.89
1002	XY	55.66

CASE-1: Inserting single record

```
SQL> INSERT INTO student
        VALUES(1001, 'ABC', 67.89);
```

Output:

1 row created.

SQL> INSERT INTO student
VALUES(1002, 'XY', 55.66);
Output:
1 row created.

=> 2 records inserted in INSTANCE [RAM]

SQL> COMMIT;

=> when COMMIT command is executed,
then data in INSTANCE (RAM) will be moved to DB (Hard Disk).

SQL> SELECT * FROM student;
Output:

SID	SNAME	AVRG
1001	ABC	67.89
1002	XY	55.66

CASE-2: Inserting limited column values

STUDENT

SID	SNAME	AVRG
1003	A	

INSERT INTO student VALUES(1003, 'A');
Output:
ERROR: not enough values

INSERT INTO student(sid, sname) VALUES(1003, 'A');
Output:
1 row created

SID	SNAME	AVRG
1004	B	

INSERT INTO student(sname, sid)
VALUES('B', 1004);
Output:
1 row created.

SID	SNAME	AVRG
1005		89.23

INSERT INTO student(avrg, sid) VALUES(89.23, 1005);

NOTE:
For any data type default value is NULL.

CASE-3: Inserting multiple records using parameters

NOTE:

- In ORACLE SQL,
  PARAMETER concept is used to read the value.

Syntax:
&<text>

Examples:
&sid
Output:
Enter value for sid: 1234

&A
Output:
Enter value for A: 50

&firstnum
Output:

Enter value for firstnum: 30

5001	ABCD	45.23

SQL> INSERT INTO student VALUES(&sid, '&sname', &avrg);
Output:
Enter value for sid: 5001
Enter sname: ABCD
Enter avrg: 45.23
INSERT INTO student VALUES(&sid, '&sname', &avrg)
INSERT INTO student VALUES(5001, 'ABCD', 45.23)
1 row created.

SQL> /
Output:
Enter value for sid: 5002
Enter sname: XYZ
Enter avrg: 56.78

SQL> /
Output:
Enter value for sid: 5003
Enter sname: AB
Enter avrg: 77.23

SQL> COMMIT;

SQL> SELECT * FROM student;

NOTE:

/	RUN	Runs recent query which is in the memory

/ (or) R[UN]

Example-2:

EMPLOYEE

EMPID	ENAME	ECITY	SAL	DOJ
7001	ABC	HYD	25000.00	25-DEC-23
7002	XY	MUM	20000.00	17-FEB-20
7003	A	DLH	30000.00	Today's date

```
CREATE TABLE employee
(
 empid NUMBER(4),
 ename VARCHAR2(10),
 ecity CHAR(3),
 sal NUMBER(8,2),
 doj DATE
);
```

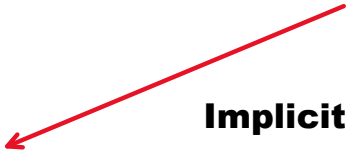
Output:

Table created.

7001	ABC	HYD	25000.00	25-DEC-23

```
INSERT INTO employee
VALUES(7001, 'ABC', 'HYD', 25000, '25-DEC-2023');
```

string



DOJ DATE

25-DEC-23 date

Implicit Conversion

NOTE:

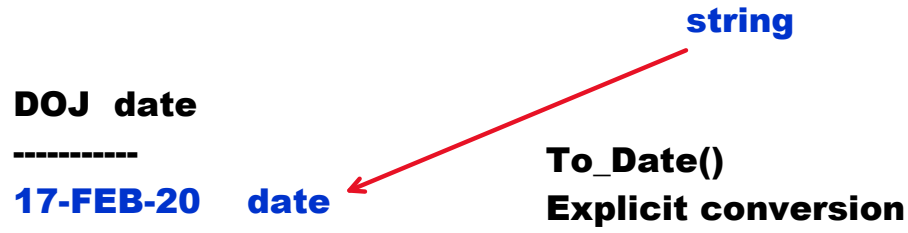
- **Don't depend on Implicit Conversion.**
  It degrades the performance. That is why do

explicit conversion.

- For explicit conversion ORACLE SQL provides conversion functions [built-in functions]:
  To_char()
  To_Date()
  To_Number() ...etc

7002	XY	MUM	20000.00	17-FEB-20

```
INSERT INTO employee
VALUES(7002, 'XY', 'MUM', 20000, to_date('17-FEB-2020'));
```



7003	A	DLH	30000.00	Today's date

```
INSERT INTO employee
VALUES(7003, 'A', 'DLH', 30000, sysdate);
```

NOTE:
- Sysdate is built-in function. It returns current system date.

```
COMMIT;

SELECT * FROM employee;
```

EMPID	ENAME	ECITY	SAL	DOJ
1001	A		15000	

```
INSERT INTO employee(empid, ename, sal)
VALUES(1001, 'A', 15000);
```

To see table structure:

Syntax:

DESC[RIBE] <table_name>

Example:

SQL> DESC student

Output:

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

To see tables list which are created by a user:

User_tables:

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

SQL> DESC user_tables

SQL> SELECT table_name
FROM user_tables;

Output:

TABLE_NAME

STUDENT
EMPLOYEE

Example-3:

ATTENDANCE

EMPID	LOGIN_DATE_TIME
1234	15-JUL-25 10.30.0.0 AM
1235	15-JUL-25 02.20.0.0 PM
1236	Current system date and time

ATTENDANCE

EMPID	LOGIN_DATE_TIME

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

Output:

Table created.

1234	15-JUL-25 10.30.0.0 AM

```
INSERT INTO attendance
VALUES(1234, '15-JUL-2025 10.30.0.0 AM');
```

string

Implicit conversion

login_date_time TIMESTAMP

15-JUL-25 10.30.0.0 AM timestamp

NOTE:
Implicit conversion degrades the performance.

1235	15-JUL-25 02.20.0.0 PM

```
INSERT INTO attendance
VALUES(1235, to_timestamp('15-JUL-2025 02.20.0.0 PM'));
```

string

To_timestamp()
Explicit conversion

login_date_time TIMESTAMP

15-JUL-2025 02.20.0.0 PM timestamp

1236	Current system date and time

```
INSERT INTO attendance
VALUES(1236, systimestamp);
```

NOTE:
Systimestamp:

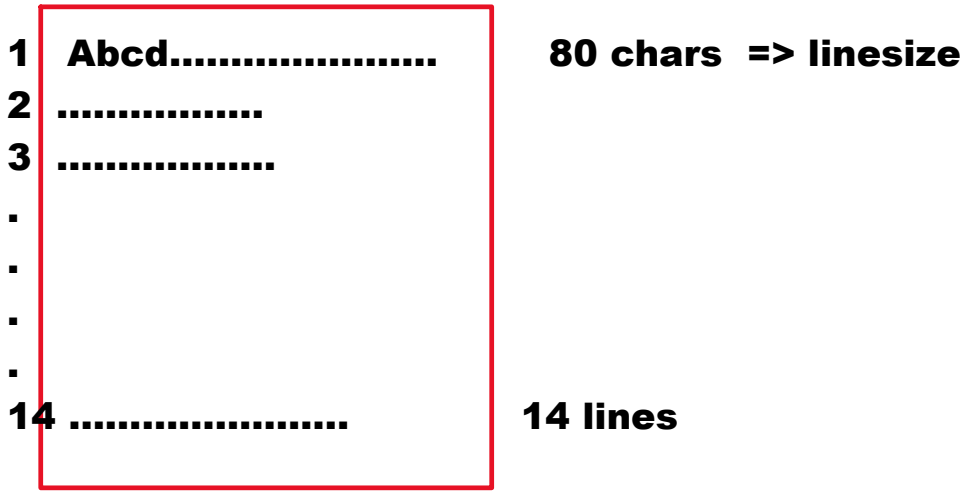
- **It is a built-in function.**
- **It returns current system date and time.**

```
COMMIT;

SELECT * FROM attendance;
```

Setting Page Size and Line Size

Thursday, July 17, 2025 9:40 AM

Setting Page Size and Line Size:



PAGESIZE	14
LINESIZE	80

SQL> SHOW PAGESIZE
Output:
PAGESIZE 14

SQL> SHOW LINESIZE
Output:
LINESIZE 80

Setting Linesize:

Syntax:
SET LINES[IZE] <value>

Example:
SQL> SET LINES 200

Setting Pagesize:

Syntax:

```
SET PAGES[IZE] <value>
```

Example:

```
SQL> SET PAGES 300
```

NOTE:

```
SET LINES 200
SET PAGES 300        (or)        SET LINES 200 PAGES 300
```

Column Alias

Thursday, July 17, 2025 10:12 AM

Column Alias:

- **Alias => another name / alternative name**
- **It is used to change column heading in output.**

Syntax:

<column> [AS] <column_alias>

Example:

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

A	B
SMITH	800

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

Editing Query

Monday, July 21, 2025 9:34 AM

Editing Query:

Syntax:

ED[IT]

- SQL> ED

- Displays editor with recent query.
- Edit it.
- Don’t write ; at ending.
- Save it [File => Save].
- Close the editor.

SQL> /

NULL

Tuesday, July 22, 2025 10:21 AM

STUDENT

SID	SNAME	M1 NUMBER(3)
1001	A	70
1002	B	0
1003	C	68
1004	D	

Max marks: 100

NULL

Unable insert ABSENT

EMPLOYEE

EMPID	ENAME	SAL
1001	A	12000
1002	B	15000
1003	C	

NULL

Sal value is unknown

NULL:

- **NULL means empty (or) blank.**
- **NULL is not equals to 0 or space.**

- **When we are unable to insert the value or when value is unknown or when value is optional we insert NULL.**

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

  Example:

  SELECT 100+200 FROM dual;

  Output:

  300

  SELECT 100+200+null FROM dual;

  Output:

  NULL

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

  NULL=NULL FALSE
  NULL!=NULL FALSE

DRL

Wednesday, July 16, 2025 9:53 AM

DRL / DQL:

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

- Retrieve => getting back => opening existing data.
- Query => request

- It deals with data retrievals.

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

SELECT:

- SELECT command is used to retrieve the data from table.
- Using SELECT command we can select:
  - All columns, All rows
  - Specific columns, All rows
  - All columns, Specific rows
  - Specific columns, Specific rows

Syntax:

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

SQL	ENGLISH
Queries	SENTENCES
Clauses	WORDS

NOTE:

CLAUSE:

- **CLAUSE is a part of query.**
- **Every query is made up of with clauses.**
- **Every clause has specific purpose.**

SELECT:

- **SELECT clause is used to specify column list.**

FROM:

- **FROM clause is used to specify table name.**

WHERE:

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

Examples on SELECT:

CASE-1: All columns, All rows

Display all columns and all rows from student table:

```
SELECT *
FROM student;
```

*	All columns

Above query will be rewritten by ORACLE as following:

```
SELECT sid, sname, avrg
FROM student;
```

*** (star) will be replaced with all column names in table structure order**

CASE-2: Specific columns, All rows

Display all students names and their avrg marks:

```
SELECT sname, avrg
FROM student;
```

CASE-3: All columns, Specific rows

Display the student record whose sid is 5001:

```
SELECT *
FROM student
WHERE sid=5001;
```

CASE-4: Sepcific columns, Specific rows

Display student name and avrg marks of sid 5001:

```
SELECT sname, avrg
FROM student
WHERE sid=5001;
```

All columns	SELECT *
Specific Columns	SELECT sname, avrg
All rows	Don't write WHERE condn
Specific rows	WHERE sid=5001

Operators in ORACLE SQL:

Operator:

- **Operator is a symbol.**
- **Operator is used to perform operations like**

arithmetic or logical operations.

OREACLE SQL provides following operators:

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

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

ORACLE SQL provides following Arithmetic Operators:
+ - * /

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;

NOTE:

- To give column alias in multiple words enclose column alias in double quotes.
  It maintains case also.

Example:

Calculate TA, HRA, TAX and GROSS salary of all emps.
10% on sal => TA
20% on sal => HRA
5% on sal => TAX (deducting)
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:

STUDENT

SID	SNAME	M1	M2	M3
1001	A	70	90	80
1002	B	56	78	45

Calculate total marks and avrg marks of all students:

SID	SNAME	TOTAL	AVRG

Find experience of all emps:

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

Relational Operators / Comparison Operators:

Syntax:

```
<column> <relational operator> <value>
```

Example:

```
WHERE sid = 5001
```

```
SID         WHERE sid = 5001
-----       ----------------------------
4998                4998 = 5001   F
4999                4999 = 5001   F
5000                5000 = 5001   F
5001                5001 = 5001   T
```

ORACLE SQL provides following Relational operators:

> < >= <= = != / <> / ^=
equals not equals

Examples on Relational Operators:

Display all managers records:

ENAME	JOB	SAL
	MANAGER	

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

Output:
No rows selected

JOB	WHERE job='manager'
ANALYST	ANALYST=manager FALSE
CLERK	CLERK = manager FALSE
MANAGER	MANAGER =manager FALSE

When all conditions are FALSE, we get output as "No rows selected."

NOTE:

- **SQL is not case sensitive language. But, String comparison is case sensitive.**
  MANAGER =manager FALSE

```
SELECT ename, job, sal
FROM emp
WHERE job='MANAGER';
--displays managers records
```

Display the emp record whose name is BLAKE:

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

Display the emp record whose empno is 7521:

```
SELECT *
FROM emp
WHERE empno=7521;
```

Display the emp records whose sal is more than 2500:

ENAME	SAL

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

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 salary is 1250 or less:

ENAME	SAL

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

Display the emp records who are working in 20th dept:

ENAME	SAL	DEPTNO
		20

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

Display all emp records except 20th dept employees:

```
SELECT ename, sal, deptno
FROM emp
WHERE deptno!=20
ORDER BY deptno ASC;
```

Display all emp records except managers:

ENAME	JOB	SAL

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

NOTE:

CALENDAR order is ASCENDING ORDER.

ASC

1-JAN-2023
2-JAN-2023
3-JAN-2023
.
.
31-DEC-2023
1-JAN-2024
.
31-DEC-2024

Display the emp records who joined after 1981:

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

> '31-DEC-1981'

(or)

2-JAN-1982
.
.

(or)

>='1-JAN-1982'

ENAME	HIREDATE

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

Display the emp records who joined before 1981:

.
.
30-DEC-1980
31-DEC-1980
1-JAN-1981

< '1-JAN-1981'
(or)
<='31-DEC-1980'

ENAME	HIREDATE

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

Display the emp records who joined after april 1981:

30-APR-1981
1-MAY-1981
.
.

> '30-APR-1981'

ENAME	HIREDATE

```
SELECT ename, hiredate
FROM emp
WHERE hiredate>'30-APR-1981'
ORDER BY hiredate ASC;
```

Logical operators:

- Logical operators are used to perform logical operations like logical AND, logical OR, ...
- ORACLE SQL provides following Logical Operators:
  - AND
  - OR
  - NOT

AND:

- It is used to perform logical AND operations.

Syntax:

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

OR:

- It is used to perform logical OR operations.

Syntax:

```
<condition1> OR <condition2>
```

AND	All conditions should be satisfied
OR	At least 1 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:

ENAME	JOB	SAL

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

JOB	WHERE job='MANAGER' AND job='CLERK'		
SALESMAN	F	✗	F
MANAGER	T	F	F
CLERK	F	✗	F
ANALYST	F	✗	F

Output:
No rows selected

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

Output:
Displays all managers and clerks

JOB	WHERE job='MANAGER' OR job='CLERK'		
SALESMAN	F	F	F
MANAGER	T	✓	T

SALESMAN	F	F	F
MANAGER	T	✗	T
CLERK	F	T	T
ANALYST	F	F	F

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

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

Display the emp records whose salary is 2000 or more and 3000 or less [sal b/w 2000 to 3000]:

ENAME	SAL

```
SELECT ename, sal
FROM emp
WHERE sal>=2000 AND sal<=3000;
```

SAL	WHERE sal>=2000 AND sal<=3000		
2500	T	T	T
5000	T	F	F
1500	F	✗	F
2800	T	T	T

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

ENAME	SAL	DEPTNO

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

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

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

Display the managers records who are earning more than 2500:

ENAME	JOB	SAL

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

Display the managers records who joined after april 1981:

ENAME	JOB	SAL	HIREDATE

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

NOTE:

All T	AND
Min 1 T	OR

Display the CLERKs and SALESMEN whose salary is more than 1250:

ENAME	JOB	SAL

```
SELECT ename, job, sal
FROM emp
WHERE (job='CLERK' OR job='SALESMAN') AND sal>1250;
```

NOT:

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

Syntax:

```
NOT(<condition>)
```

Truth Table:

Condition	NOT(Condition)
T	NOT(T) => F
F	NOT(F) => T

Examples on NOT operator:

Display all emp records except managers:

ENAME	JOB	SAL

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

JOB	WHERE NOT(job='MANAGER')
MANAGER	NOT(T) => F
CLERK	NOT(F) => T

ANALYST	NOT(F) => T
MANAGER	NOT(T) => F

Display all emp records except 30th dept emps:

ENAME	SAL	DEPTNO

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

Concatenation Operator:

- Symbol: ||
- Concatenate => combine
- It is used to combine 2 strings.

Syntax:

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

Examples on Concatenation operator:

Display output as following:
SMITH works as CLERK
ALLEN works as SALESMAN

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

Display output as following:
SMITH joined on 17-DEC-80
ALLEN joined on 20-FEB-81

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

Display output as following:
SMITH works as CLERK and earns 800
ALLEN works as SALESMAN and earns 1600

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

Example:

EMPLOYEE

FNAME	LNAME
RAVI	TEJA
KIRAN	KUMAR
SAI	KRISHNA

```
SELECT fname || ' ' || lname AS ename
FROM employee;
```

Output:

```
ENAME
---------------
RAVI TEJA
KIRAN KUMAR
SAI KRISHNA
```

IN:

Syntax:

```
<column> IN(<value list>)
```

Example:

```
WHERE sal IN(1250, 3000, 5000)
```

```
SAL
---------
```

2500 2500 not in list => F
3000 3000 is in list => T

- IN operator is used to compare column value with list of values.
- If column value is in list then condition is TRUE.
- If column value not in list then condition is FALSE.
- It avoids multi equality conditions using OR.

Examples on IN:

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

ENAME	SAL
	1250
	3000
	5000

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

(or)

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

Display all managers and clerks records:

ENAME	JOB	SAL

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

JOB	WHERE job IN('MANAGER', 'CLERK')

ANALYST	ANALYST	FALSE
MANAGER	MANAGER	TRUE
SALESMAN	SALESMAN	FALSE
CLERK	CLERK	TRUE

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 ALLEN, BLAKE, JAMES:

```
SELECT *
FROM emp
WHERE ename IN('ALLEN', 'BLAKE', 'JAMES');
```

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)
ORDER BY deptno ASC;
```

Display all emp records except managers and clerks:

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

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

ANALYST	ANALYST not in list TRUE
MANAGER	MANAGER is in list FALSE
SALESMAN	SALESMAN not in list TRUE
CLERK	CELRK is in list FALSE

Display all emp records except deptno 10 and 30 emps:

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

BETWEEN AND:

Syntax:

```
<column> BETWEEN <lower> AND <upper>
```

- It is used to compare column value with range of values.
- If column value is in range, condition is TRUE.
- If column value is not in range, condition is FALSE.

Examples on BETWEEN AND:

Display the emp records whose salary is between 2000 to 3000:
(or) [whose sal is 2000 or more and 3000 or less]

ENAME	SAL

```
SELECT ename, sal
FROM emp
WHERE sal BETWEEN 2000 AND 3000
ORDER BY sal ASC;
```

SAL	WHERE sal BETWEEN 2000 AND 3000
2500	T

4000	F
1250	F
3000	T
2000	T
2800	T

Display the emp records whose empnos are between 7800 to 7900:

```
SELECT *
FROM emp
WHERE empno BETWEEN 7800 AND 7900;
```

Display the emp records who joined in 1982:

ENAME	HIREDATE

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

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

Display the emp records whose salaries are not between 2000 and 3000:

```
SELECT ename, sal
FROM emp
WHERE sal NOT BETWEEN 2000 AND 3000;
```

SAL	WHERE sal NOT BETWEEN 2000 AND 3000
1500	T
2500	F

4000	T
2800	F

Display the emp records who are not joined in 1981:

ENAME	HIREDATE

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

NOTE:

IN	To compare column value with LIST
BETWEEN AND	To compare column value with RANGE

LIKE:

Syntax:

```
<column> LIKE <text pattern>
```

- It is used to compare column value with text pattern.
- To specify text pattern ORACLE SQL provides following wildcard chars:

Wildcard char	Purpose
_	Replaces 1 char
%	Replaces 0 or any

Examples on LIKE:

%	0 or any
_	1 char

Display the emp records whose names Are started with 'S':

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

ENAME	WHERE ename LIKE 'S%'
SMITH	T
WARD	F
SYMONDS	T
ALLEN	F

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 are having 4 letters:

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

Display the emp records who are getting 3 digit salary:

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

Display the emp records who joined in DECEMBER month:

ENAME	HIREDATE

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

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 _ (underscore):

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

_	Wildcard char
_	Normal char _

IS NULL:

- It is used to compare column value with NULL.

  Syntax:

  ```
  <column> IS null
  ```

Examples:

Display the emp records who are not getting commission:

ENAME	SAL	COMM

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

Display the emp records who are getting commission:

ENAME	SAL	COMM

```
SELECT ename, sal, comm
FROM emp
WHERE comm IS not null;
```

UPDATE

Wednesday, July 23, 2025 9:34 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>=<new_value> [, <column>=<new_value>, ....]
[WHERE <condition>];
```

CASE-1: Modifying single value of single record

Increase 2000 rupees salary to 7499 employee:

```
UPDATE emp
SET sal=sal+2000
WHERE empno=7499;

COMMIT;
```

CASE-2: Modifying multiple values of single record

Modify job value as MANAGER, sal as 6000 to the employee whose empno is 7369:

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

CASE-3: Modifying Specific group of records

Increase 20% on sal to all managers:

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

Increase 10% on sal to the emps who are working in deptno 30:

```
UPDATE emp
SET sal=sal+sal*0.1
WHERE deptno=30;
```

CASE-4: Modifying all records

Increase 15% on sal to all emps:

```
UPDATE emp
SET sal=sal+sal*0.15;
```

Examples on UPDATE:

Transfer all deptno 10 emps to deptno 20:

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

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

```
UPDATE emp
SET sal=sal+sal*0.2, comm=comm+comm*0.1
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 and 7698:

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

NOTE:

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

Increase 20% on sal to the emps who joined in 1982:

```
UPDATE emp
SET sal=sal+sal*0.2
WHERE hiredate BETWEEN '1-JAN-1982' AND '31-DEC-1982';
```

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 12% on sal to the emps who are having more than 43years experience:

```
UPDATE emp
SET sal=sal+sal*0.12
WHERE trunc((sysdate-hiredate)/365)>43;
```

Assignment:

ACCOUNTS

ACNO	BALANCE
1234	80000
1235	20000

Withdraw 10000 rupees from 1234 account:

Deposit 30000 rupees to 1235 account:

Transfer 10000 rupees from 1234 a/c to 1235 a/c:

UPDATE from a/c
UPDATE to a/c

Example:

EMPLOYEE10

EMPID	ENAME	SAL	TA	HRA	TAX	GROSS
1234	A	12000				
1235	B	15000				

10% on sal => TA
20% on sal => HRA
5% on sal => TAX
GROSS = sal+TA+HRA-TAX

EMPLOYEE10

EMPID	ENAME	SAL	TA	HRA	TAX	GROSS

```
CREATE TABLE employee10
(
  empid NUMBER(4),                                    100000.00
  ename VARCHAR2(10),
  sal NUMBER(8,2),
  TA NUMBER(8,2),
  HRA NUMBER(8,2),
  TAX NUMBER(8,2),
  GROSS NUMBER(8,2)
);
```

EMPID	ENAME	SAL	TA	HRA	TAX	GROSS

1234	A	12000				

```
INSERT INTO employee10(empid, ename, sal)
VALUES(1234, 'A', 12000);
```

EMPID	ENAME	SAL	TA	HRA	TAX	GROSS
1235	B	15000				

```
INSERT INTO employee10
VALUES(1235, 'B', 15000, null, null, null, null);

COMMIT;

SELECT * FROM employee10;
```

Calculate TA, HRA, TAX and GROSS:

```
UPDATE employee10
SET TA=sal*0.1, HRA=sal*0.2, TAX=sal*0.05;

UPDATE employee10
SET gross=sal+TA+HRA-TAX;

COMMIT;

SELECT * FROM employee10;
```

DELETE

Thursday, July 24, 2025 9:46 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**

Syntax:

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

Examples on DELETE:

CASE-1: Deleting single record

Delete the emp record whose empno is 7876:

```
DELETE FROM emp
WHERE empno=7876;

COMMIT;
```

CASE-2: Deleting specific group of records

Delete all managers records:

```
DELETE FROM emp
WHERE job='MANAGER';
```

Delete all deptno 30 emps:

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

Delete all deptno 10 and 20 emps:

```
DELETE FROM emp
WHERE deptno IN(10,20);
```

Delete the emp records who joined in 1982:

```
DELETE FROM emp
WHERE hiredate BETWEEN '1-JAN-1982' AND '31-DEC-1982';
```

Delete the emp records whose annual salary is more than 35000:

```
DELETE FROM emp
WHERE sal*12>35000;
```

Delete the emp records who are having more than 43years experience:

```
DELETE FROM emp
WHERE trunc((sysdate-hiredate)/365)>43;
```

CASE-3: Deleting all records

Delete all rows from emp table:

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

ALTER

Thursday, July 24, 2025 10:02 AM

TABLE = STRUCTURE (cols) + DATA (rows)

EMPLOYEE

EMPID	ENAME	SAL
1001	A	15000
1002	B	12000
1003	C	20000
1004	D	18000

Table structure (columns)

Table data (rows)

ALTER:

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

Syntax:

```
ALTER TABLE <table_name> [ADD(<column> <data_type>, ... , ...)]
                         [RENAME COLUMN <old_name> TO <new_name>]
                         [DROP COLUMN <column_name>]
                         [DROP(<column_list>)]
                         [MODIFY(<column> <new_data_type>(<new_size>), ...)];
```

Example:

STUDENT10

SID	SNAME

```
CREATE TABLE student10
(
  sid NUMBER(4),
```

```
  sname VARCHAR2(10)
);
```

Output:

Table created.

DESC student10

Output:

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

Add m1 column:

```
ALTER TABLE student10 ADD m1 NUMBER(3);
```

Output:

Table Altered.

DESC student10

Output:

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

Add m2 and m3 columns:

```
ALTER TABLE student10
ADD(m2 NUMBER(3), m3 NUMBER(3));
```

Output:

Table Altered.

DESC student10

Output:

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

Rename M3 to MATHS:

```
ALTER TABLE student10
```

RENAME COLUMN m3 TO maths;
Output:
Table Altered.

DESC student10
Output:

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

Drop MATHS column:

ALTER TABLE student10 DROP COLUMN maths;
(or)
ALTER TABLE student10 DROP(maths);
Output:
Table Altered.

DESC student10
Output:

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

Drop m1 and m2 columns:

ALTER TABLE student10 DROP(m1, m2);
Output:
Table Altered.

DESC student10
Output:

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

Modify sname field size from 10 to 20:

ALTER TABLE student10 MODIFY sname VARCHAR2(20);
Output:
Table Altered.

DESC student10
Output:

NAME	TYPE
SID	NUMBER(4)
SNAME	VARCHAR2(20)

Modify SID data type to CHAR(7):

ALTER TABLE student10 MODIFY sid CHAR(7);
Output:
Table Altered.

DESC student10
Output:

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

NOTE:
To modify data type column must be empty.

TCL:

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

- Transaction:
  Transaction is a series of actions.

Examples:
Withdraw, Deposit, Fund transfer, placing order

Example:

FUND TRANSFER TRANSACTION

ACCOUNTS

ACNO	BALANCE
1234	80000
1235	20000

Transfer RS 10000/- from 1234 a/c to 1235:

BEGIN TRANSACTION
- Sufficient funds available? [SELECT]
- Update FROM a/c balance [UPDATE]
- Update TO a/c balance [UPDATE]

END TRANSACTION

Transaction Rule:

Every transaction must be successfully finished or cancelled.

If transaction is successful, save it (COMMIT).
If transaction is unsuccessful, cancel it (ROLLBACK).

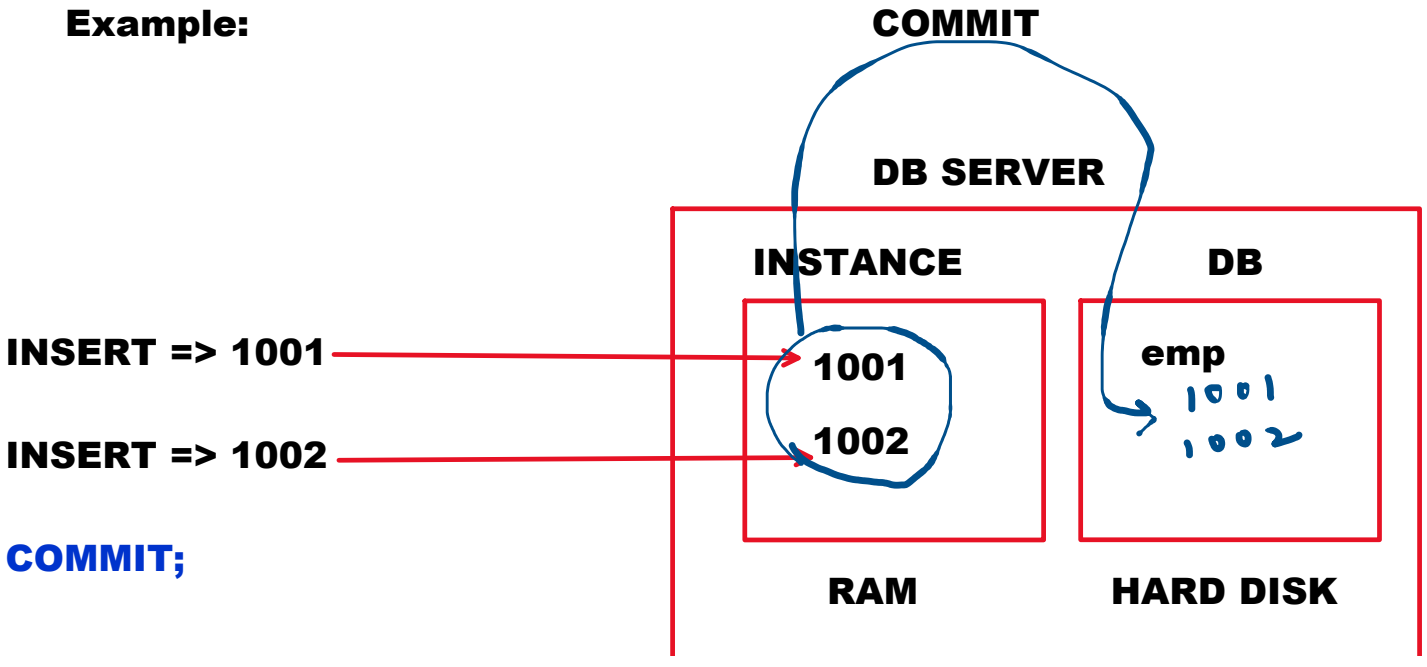
ORACLE SQL provides following TCL commands:

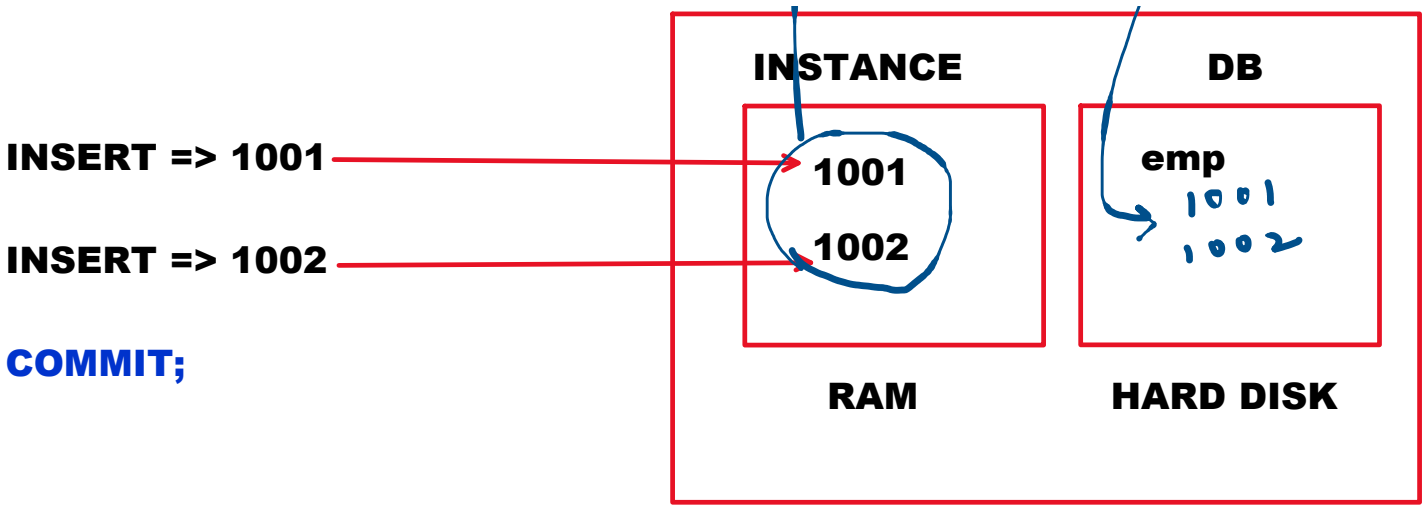
- **COMMIT**
- **ROLLBACK**
- **SAVEPOINT**

COMMIT [save]:

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

Example:





ROLLBACK:

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

Syntax:

ROLLBACK [TO <savepoint_name>];

Example on COMMIT and ROLLBACK:

T1

F1 INT
1
2

```
CREATE TABLE t1(f1 INT);      --committed

INSERT INTO t1 VALUES(1);
INSERT INTO t1 VALUES(2);
COMMIT;                              --committed

INSERT INTO t1 VALUES(3);
```

```
INSERT INTO t1 VALUES(4);

SELECT * FROM t1;
Output:
F1
--
1
2
3
4

ROLLBACK;

SELECT * FROM t1;
Output:
F1
--
1
2
```

SAVEPOINT:

- SAVEPOINT command is used to specific point for ROLLBACK.

```
Syntax:
   SAVEPOINT <savepoint_name>;
```

Example:

T2

F1 INT
1

2

```
CREATE TABLE t2(f1 INT);    --committed

INSERT INTO t2 VALUES(1);
INSERT INTO t2 VALUES(2);

SAVEPOINT p1;

INSERT INTO t2 VALUES(3);
INSERT INTO t2 VALUES(4);

SAVEPOINT p2;

INSERT INTO t2 VALUES(5);
INSERT INTO t2 VALUES(6);

SELECT * FROM t2;
Output:
F1
----
1
2
3
4
5
6

ROLLBACK TO p2;

SELECT * FROM t2;
Output:
F1
----
1
2
```

3
4

NOTE:

COMMIT	**To save the transaction**
ROLLBACK	**To cancel the transaction**
SAVEPOINT	**To set specific point for ROLLBACK**

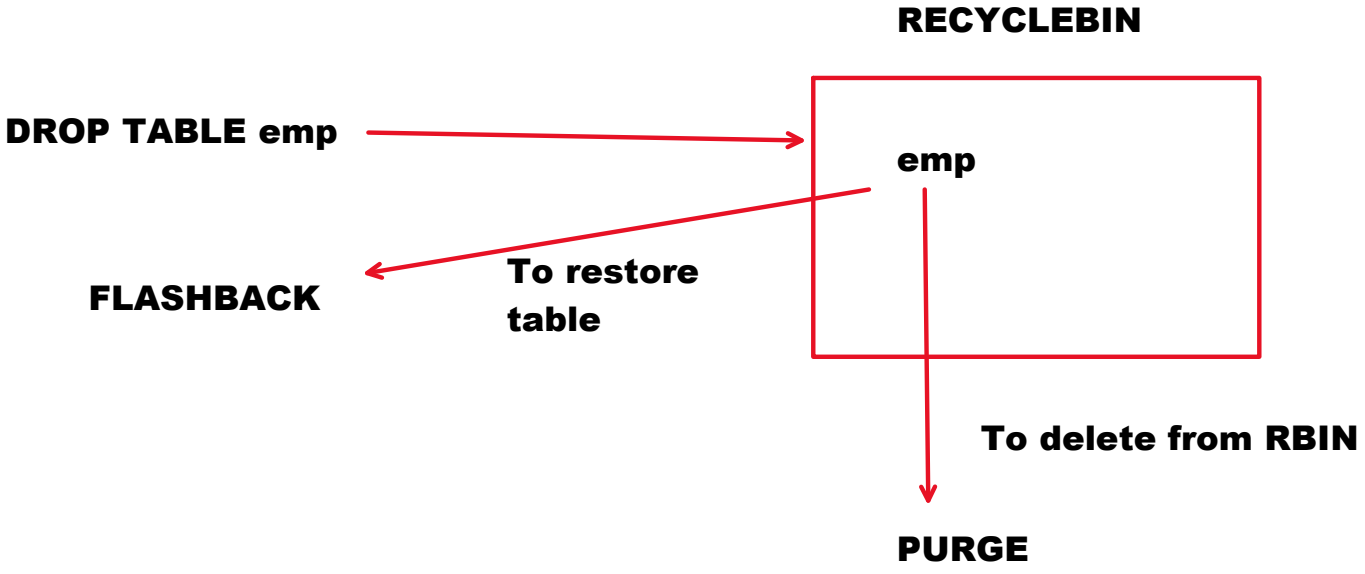
DROP, FLASHBACK and PURGE

Friday, July 25, 2025 10:31 AM

DROP, FLASHBACK and PURGE:

In ORACLE 10g, RECYCLEBIN feature added.

FLASHBACK, PURGE commands introduced in ORACLE 10g version.
These are related to RECYCLEBIN.



DROP:

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

Syntax:

DROP TABLE <table_name> [PURGE];

Examples:

```
DROP TABLE student;
--table will be moved to RECYCLEBIN
```

To see recyclebin:

SHOW RECYCLEBIN

Output:

ORIGINAL_NAME	RECYCLEBIN_NAME	DROP_DATE_TIME
STUDENT	..	..

DROP TABLE employee PURGE;
--deletes employee table permanently

FLASHBACK:

- It is used to restore the dropped table.

Syntax:

```
FLASHBACK TABLE <table_name>/"<rbin_name>"
TO BEFORE DROP;
```

Example:

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

PURGE:

- It is used to delete the table from RECYCLEBIN.
- It deletes table permanently. Again we cannot recollect it.

Syntax:

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

Example:

```
DROIP TABLE employee;
--employee table will be moved to RECYCLEBIN

PURGE TABLE employee;
--deletes from RECYCLEBIN
--table deleted permanently
```

DROP TABLE employee;
PURGE TABLE employee; **(Or)** **DROP TABLE employee PURGE;**

To empty RECYCLEBIN:

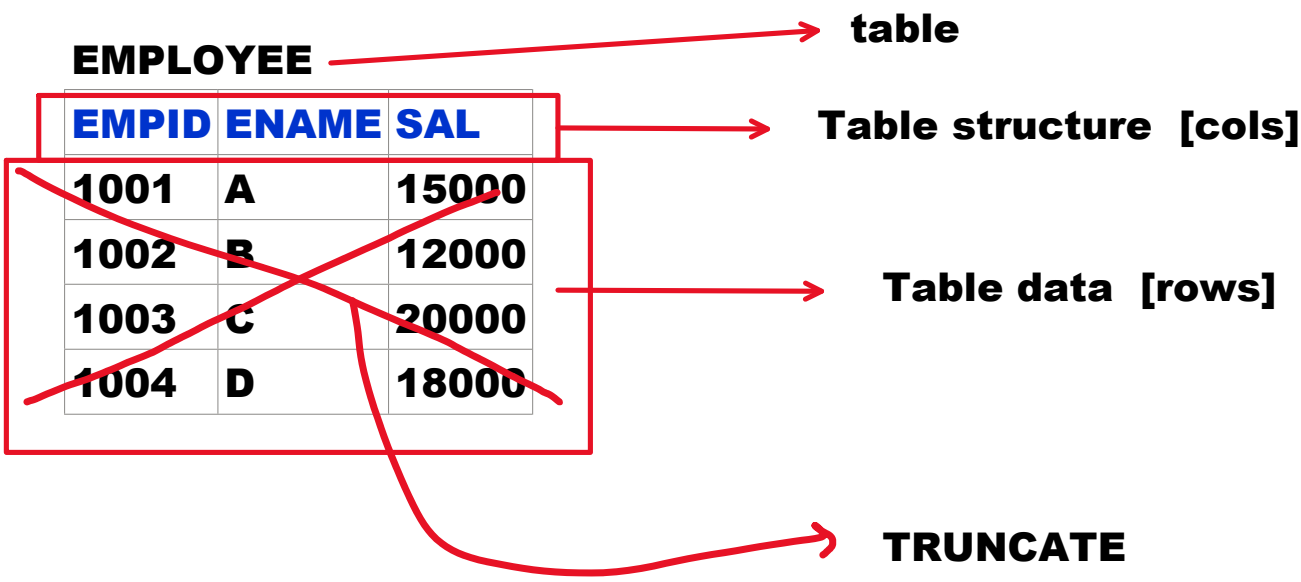
RBIN
T1
T2
.
.
T50

PURGE RECYCLEBIN;
Output:
RECYCLEBIN purge

NOTE:
RECYCLEBIN will not work for SYSTEM user by default.

TRUNCATE

Saturday, July 26, 2025 9:47 AM



TRUNCATE:

- **TRUNCATE command is used to delete entire table data [rows] with good performance.**
- **It deletes all rows permanently. Again we cannot recollect them.**
- **It deletes all rows only. It does not delete table structure.**

Syntax:

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

Example:

```
TRUNCATE TABLE employee10;
--deletes all rows

DESC employee10
Output:
NAME        TYPE
-----------------------------
EMPID          ..
```

....

SELECT * FROM employee10;

Output:

No rows selected.

RENAME

Saturday, July 26, 2025 9:55 AM

RENAME:

It is used to rename the Table.

Syntax:

RENAME <old_name> TO <new_name>;

Example:

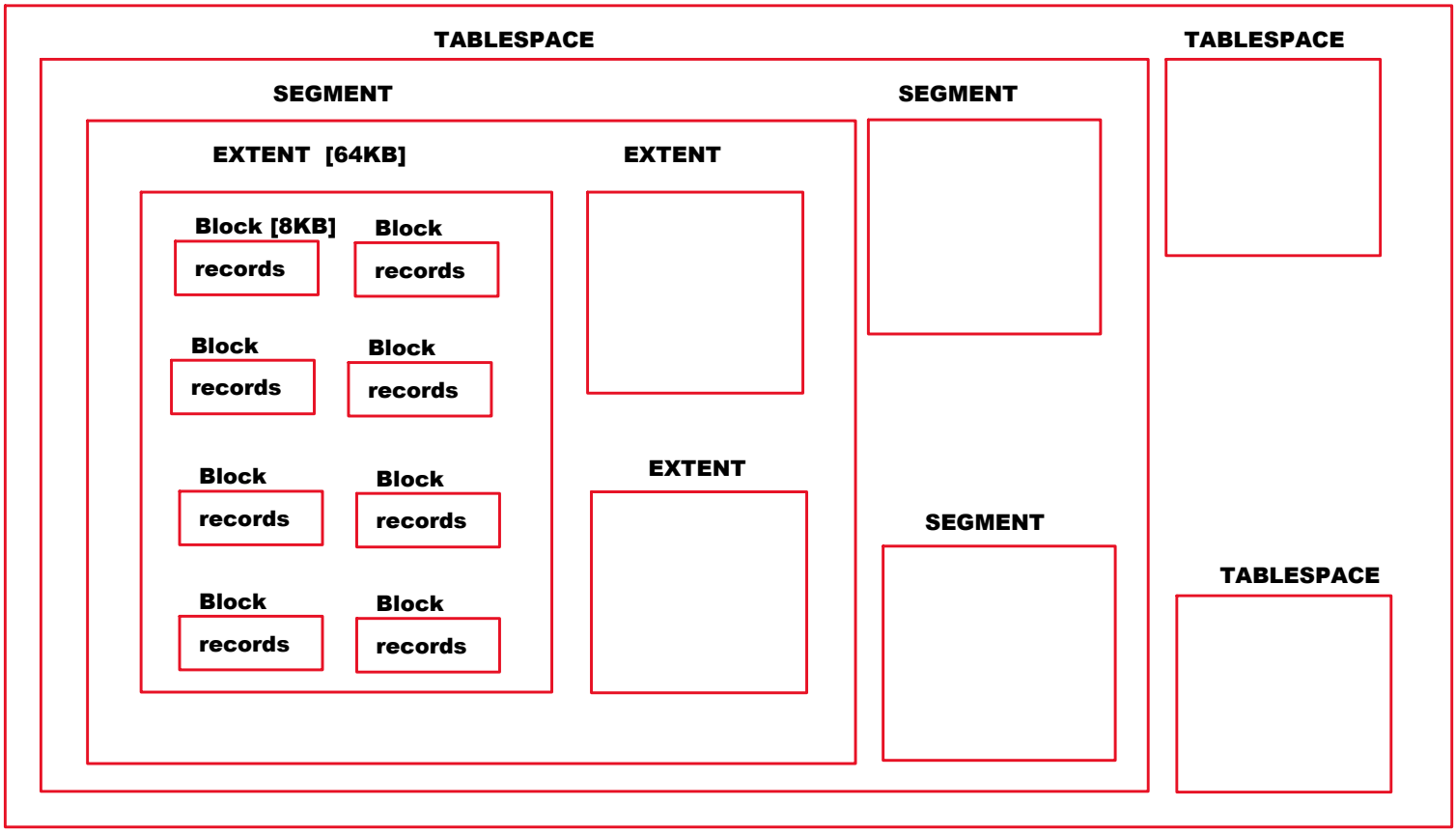
RENAME attendance TO a;

TRUNCATE TABLE emp; --deletes all rows
DELETE FROM emp; --deletes all rows

Differences b/w TRUNCATE and DELETE:

TRUNCATE	DELETE
• It is DDL command.	• It is DML command.
• Using TRUNCATE, We cannot delete 1 row or Specific group of rows. Only We can delete all rows.	• Using DELETE, • We can delete 1 row or specific group of rows or all rows.
• WHERE clause cannot be used here.	• WHERE clause can be used here.
• It is auto committed.	• It is not auto committed.
• It cannot be rolled back.	• It can be rolled back.
• It is faster.	• It is slower.
• It deletes block by block [Page by page].	• It deletes row by row.

DATABASE



DATABASE
TABLESPACES
SEGMENTS
EXTENTS
BLOCKS
RECORDS

NOTE:

- **All DDL commands are auto committed.**
- **All DML commands are not auto committed.**

- **DDL Command = DDL command + COMMIT**
  CREATE = CREATE + COMMIT
  ALTER = ALTER + COMMIT
  TRUNCATE = TRUNCATE + COMMIT

Example:

```
CREATE TABLE t1(f1 INT);      --committed

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

CREATE TABLE t2(f1 INT);      --CREATE+COMMIT => committed

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

ROLLBACK;           --cancels 2 actions
```

SQL PLUS/
SQL DEVELOPER

ORACLE => RDBMS
DATABASE
TABLES
rows & cols

SQL => queries =>
PL/SQL => programs =>

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

EMP

EMPID	ENAME	SAL	GENDER
1001	A	15000	

DUAL

Monday, July 28, 2025 9:41 AM

DUAL:

- **DUAL is a built-in table / readymade table which is created in "SYS" schema[user].**
- **It has 1 column and 1 row.**

DUAL

DUMMY
X

SQL> DESC dual
Output:

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

SQL> SELECT * FROM dual;
Output:

D
X

Calculate 100+200:

SQL> SELECT 100+200 FROM dual;
Output:
300

Dual	1 row

SQL> SELECT 100+200 FROM emp;

Output:

300

300

300

.

.

Emp	15 rows

15 300s

- **To work with non-table data, to get 1 value as result we use DUAL".**

BUILT-IN FUNCTIONS

Saturday, July 26, 2025 10:37 AM

BUILT-IN FUNCTIONS:

- **To make our actions easier ORACLE SOFTWARE DEVELOPERS already defined some functions and placed them in ORACLE DB. These functions are called " Built-In Functions / Predefined Functions / System-defined functions / readymade functions".**

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

String Functions:

Lower()	Lpad()	Substr()
Upper()	Rpad()	Instr()
Initcap()		
	Ltrim()	Replace()
Length()	Rtrim()	Translate()
Concat()	Trim()	

NOTE:
A function can be called from SQL command like SELECT, UPDATE, INSERT, DELETE, CREATE..

Lower(), Upper(), Initcap():

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;
```

Upper():

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

Syntax:

Upper(<string>)

Example:

UPPER('raj kumar')	**RAJ KUMAR**

Initcap():

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

Syntax:

Initcap(<string>)

Examples:

Initcap('RAJ KUMAR')	**Raj Kumar**
Initcap('RAJ KUMAR VARMA')	**Raj Kumar Varma**
Initcap('RAJ')	**Raj**

Examples:

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

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

Output:

ENAME	SAL
smith	
allen	

Modify all emp names to initcap case:

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

```
ENAME
----------
SMITH
ALLEN
```

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

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

BLAKE=blake FALSE

Output:

No rows selected.

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

Output:

Displays BLAKE record

ENAME	WHERE lower(ename)='blake'
SMITH	Lower('SMITH') smith=blake FALSE
ALLEN	Lower('ALLEN') allen=blake FALSE
BLAKE	Lower('BLAKE') blake=blake TRUE

Substr():

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

Syntax:

Substr(<string> , <position> [, <no of chars>])

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

Substr('RAVI TEJA', 6)	TEJA
Substr('RAVI TEJA', 6, 3)	TEJ
Substr('RAVI TEJA', 1, 4)	RAVI

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', 6)	UMAR

Substr('RAJ KUMAR', 6, 3)	UMA
Substr('RAJ KUMAR', 1, 3)	RAJ
Substr('RAJ KUMAR', 2, 4)	AJ K

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', -5)	KUMAR
Substr('RAJ KUMAR', -4, 3)	UMA
Substr('RAJ KUMAR', -9, 3)	RAJ

Examples on Substr():

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

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

Display the emp records whose names are ended with 'S':

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

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);
```

Display the emp records whose names are started with VOWEL:

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

Display the emp records whose names are ended with 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 VOWEL:

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

Realtime scenario:

Every month BANK sends credit card bill to mail
Attach File

Open File => enter password:

Your password is:
Your name's first 4 letters
Your mobile numbers last 4 digits

Cust name: RAVI KUMAR
Mobile: 9123456789

```
Substr(cname, 1, 4) || Substr(mobile, -4, 4)
RAVI6789
```

AADHAR CARD => downloading from official website

Open file => enter password

Your password is:
Your name's first 4 letters
In DOB, year 4 digits

Example:

Generate mail ids to all emps by taking
Emp name's first 3 letters and empno's last 3 digits
as username for the domain 'tcs.com':

EMPNO	ENAME
7369	SMITH
7499	ALLEN

```
MAIL_ID
----------
```

Add mail id column to emp table:

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

Generate mail ids:

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

Lpad() and Rpad():

Pad => Fill
L => Left
R => Right

Lpad():

- **It is used to fill characters at left side.**

Syntax:

```
Lpad(<string> , <size> [, <char / chars>])
```

3rd arg default value => space

Rpad():

- **It is used to fill characters at right side.**

Syntax:

```
Rpad(<string> , <size> [, <char / chars>])
```

3rd arg default value => space

Examples:

LPAD('RAJU', 10, '*')	******RAJU
RPAD('RAJU', 10, '*')	RAJU******

Lpad('KIRAN', 8, '@')	@@@KIRAN
Rpad('KIRAN', 8, '@')	KIRAN@@@

LPAD('SAI', 10, '@#')	@#@#@#@SAI
RPAD('SAI', 10, '@#')	SAI@#@#@#@

LPAD('RAJU', 10)	6spacesRAJU
RPAD('RAJU',10)	RAJU6spaces

LPAD('A', 8, 'A')	AAAAAAAA
LPAD('X', 6, 'X')	XXXXXX
RPAD('X', 6, 'X')	XXXXXX

Example:
Display message as following:
'Amount transferred to mobile num XXXXXX6789'

Mobile: 9123456789

```
SELECT 'Amount transferred to mobile num ' ||
Lpad('X',6,'X') || Substr('9123456789', -4, 4)
FROM dual;
```

LTRIM(), RTRIM() and TRIM():

TRIM => remove
L => Left
R => Right

LTRIM():

- **It is used to remove unwanted chars from left side.**

Syntax:

```
LTRIM(<string> [, <char/chars>])
```

2nd arg, default char is SPACE

RTRIM():

- **It is used to remove unwanted chars from right side.**

Syntax:

```
RTRIM(<string> [, <char/chars>])
```

2nd arg, default char is SPACE

Examples:

LTRIM('*****RAJU*****', '*')	RAJU*****
RTRIM('*****RAJU*****', '*')	*****RAJU

LTRIM(' RAJU ')	RAJU3spaces
RTRIM(' RAJU ')	3spacesRAJU

LTRIM('@###@@@@#RAJU@###@@@@#', '@#')	RAJU@###@@@@#

LTRIM('ABBAS', 'AB')	S

TRIM():

- **Using it, we can remove unwanted chars from left side or right side or both sides.**

Syntax:

TRIM(Leading/Trailing/Both <char> FROM <string>)

Default side => BOTH
Default char => SPACE

******RAJU******

Examples:

TRIM(Leading '*' FROM '******RAJU******')	RAJU******
TRIM(Trailing '*' FROM '******RAJU******')	******RAJU
TRIM(Both '*' FROM '******RAJU******') (or) TRIM('*' FROM '******RAJU******')	RAJU

TRIM(' RAJU ')	RAJU

Length():

- **It is used to find string length.**
- **string length => number of chars in string.**

Syntax:

Length(<string>)

Examples:

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

Examples:

Display the emp records whose names are having 4 letters:

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

Display the emp records whose names are having 40 letters:

```
SELECT *
FROM emp
WHERE length(ename)=40;
```

Display the emp records whose names are having 6 letters or more:

```
SELECT *
FROM emp
WHERE length(ename)>=6;
```

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') (or) 'RAJ' || 'KUMAR' || 'VARMA'	RAJKUMARVARMA

Example:

EMPLOYEE

EMPID	FNAME	LNAME
1001	RAVI	TEJA
1002	SAI	KRISHNA

```
ENAME
----------
RAVI TEJA
SAI KRISHNA
```

```
SELECT empid,
Concat(Concat(fname, ' '), lname) AS ename
FROM employee;

(or)

SELECT empid,
Fname || ' ' || Lname AS ename
FROM employee;
```

Replace() and Translate():

Replace():

- It is used to replace search string with replace string.

Syntax:

```
Replace(<string>, <search_string>, <replace_string>)
```

Examples:

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

Translate():

- It is used to replace search char with corresponding char in replace char set.

Syntax:

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

Examples:

Replace('SAI KRISHNA', 'SAI', 'XYZ')	XYZ KRISHNA
Translate('SAI KRISHNA', 'SAI', 'XYZ')	XYZ KRZXHNY

Difference b/w replace() and translate():

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

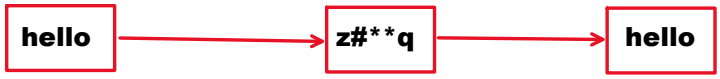
Replace('abcabcaabbccabc', 'abc', 'xyz')	xyzxyzaabbccxyz
Translate('abcabcaabbccabc', 'abc', 'xyz')	xyzxyzxxyyzzxyz

We can remove special chars.

Replace('RA@#$JU', '@#$', '')	RAJU

Using translate(), we can encrypt and decrypt the data.

Encryption => converts normal text to cipher text.
Decryption => converts cipher text to normal text.



Normal text Cipher text Normal text

ENCRYPTION DECRYPTION

Example:

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

0	1	2	3	4	5	6	7	8	9
z	@	*	$	q	b	w	!	n	y

```
SELECT ename, translate(sal, '0123456789', 'z@*$qbw!ny') AS sal
FROM emp;
```

Instr():

- It is used to search for sub string in string.
- If sub string is found, it returns position number.
- If sub string is not found, it returns 0.

Syntax:

Instr(<string>, <search_string> [, <search_position>, <occurrence>])

3rd arg, search_position default value: 1
4th arg, occurrence default value: 1

Instr('RAVI TEJA', 'TEJ', 1, 1)	6
Instr('RAVI TEJA SAI TEJA', 'TEJ', 1, 2)	15

Instr('SAI KRISHNA', 'KRISH')	5
Instr('SAI KRISHNA', 'SAI')	1
Instr('SAI KRISHNA', 'RAVI')	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', 1, 1)	3
Instr('THIS IS HIS WISH', 'IS', 4, 2)	10
Instr('THIS IS HIS WISH', 'IS', 1, 2)	6
Instr('THIS IS HIS WISH', 'IS', 5, 3)	14
Instr('THIS IS HIS WISH', 'IS', 9, 3)	0

3rd arg, search_position can be -ve.

+ve	From left side position number
-ve	From right side position number

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

Instr('THIS IS HIS WISH', 'IS', -1, 2)	10
Instr('THIS IS HIS WISH', 'IS', -8, 2)	3
Instr('THIS IS HIS WISH', 'IS', -8, 3)	0

Example:

Display the emp records whose names are having AM letters:

ENAME	WHERE Instr(ename, 'AM')>0		
SMITH	Instr('SMITH', 'AM')	0 > 0	F
ADAMS	Instr('ADAMS', 'AM')	3 > 0	T
AMAR	Instr('AMAR', 'AM')	1 > 0	T

JAMES	Instr('JAMES', 'AM')	2 > 0	T
ALLEN	Instr('ALLEN', 'AM')	0 > 0	F

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

Conversion Functions:

There are 2 types of conversions. They are:

- Implicit Conversion
- Explicit Conversion

Implicit Conversion:
If ORACLE converts implicitly from one type to another then it is called "implicit Conversion".

Example:

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

         100 + 200
        num    num
```

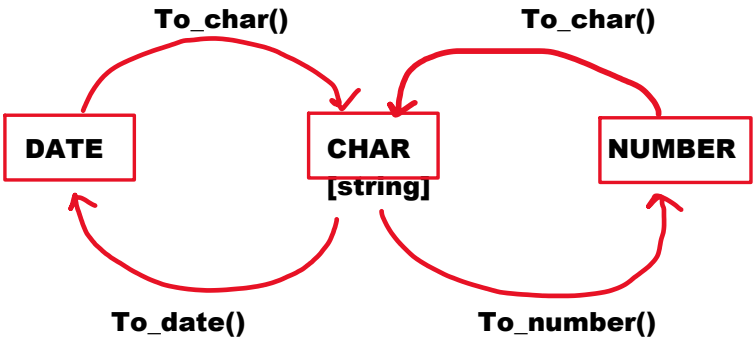
Output:
300

Explicit Conversion:
If conversion is done using function then it is called "Explicit Conversion".`

NOTE:

- ORACLE supports to implicit conversion. It degrades performance.
- To improve performance we use Explicit conversion.
- For Explicit conversion ORACLE SQL provides Following Conversion Functions:
  - To_char()
  - To_Number()

To_date()



To_char() [date to char]:

- It can be used to convert DATE to CHAR [string].
- To change date formats or
  To extract part of the date we need to convert
  DATE to CHAR.

Syntax:
To_char(<date> , <format>)

FORMAT	PURPOSE	EXAMPLE Sysdate => 31-JUL-25	OUTPUT
YYYY	Year 4 digits	To_char(sysdate, 'yyyy')	2025
YY	Year last 2 digits	To_char(sysdate, 'yy')	25
YEAR / Year	Year in words	To_char(sysdate, 'YEAR') To_char(sysdate, 'year')	TWENTY TWENTY-FIVE twenty twenty-five
MM	Month num	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')	5
DD	Day num in month	To_char(sysdate, 'DD')	31
DDD	Day num in year	To_char(sysdate, 'DDD')	31+28+31+30+31+30+31 = 212
DY	Short weekday name	To_char(sysdate, 'DY')	THU
DAY	Full weekday name	TO_char(sysdate,'DAY')	THURSDAY
HH / HH12	Hrs part in 12 hrs format		
HH24	Hrs part in 24 hrs format		
MI	Minutes		

SS	Seconds		
FF	Fractional seconds		
AM / PM	AM or PM		
Q	Quarter number 1 => jan to mar 2 => apr to jun 3 => jul to sep 4 => oct to dec	To_char(sysdate, 'Q')	3
CC	Century num	To_char(sysdate, 'CC')	21

Display current system time in 12 hrs format:

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

Display current system time in 24 hrs format:

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

NOTE:
DATE data type can hold upto seconds only.
It cannot hold fractional seconds

TIMESTAMP data type can hold fractional seconds

YYYY	MM	D	HH/HH12	Q
YY	MON	DD	HH24	CC
YEAR	MONTH	DDD	MI	
		DY	SS	
		DAY	FF	
			AM / PM	

Examples:

Display the emp records who joined in 1982:

ENAME	HIREDATE

```
SELECT ename, hiredate
```

```
FROM emp
WHERE to_char(hiredate,'yyyy')=1982;
```

Display the emp records who joined in 1980, 1982, 1983:

ENAME	HIREDATE

```
SELECT ename, hiredate
FROM emp
WHERE to_char(hiredate, 'yyyy') IN(1980, 1982, 1983);
```

Display the emp records who joined in DECEMBER month:

ENAME	HIREDATE

```
SELECT ename, hiredate
FROM emp
WHERE to_char(hiredate, 'MM')=12;
```

Display the emp records who joined in JANUARY, SEPTEMBER and DECEMBER month:

ENAME	HIREDATE

```
SELECT ename, hiredate
FROM emp
WHERE to_char(hiredate,'MM') IN(1,9,12);
```

Display the emp records who joined in 4th quarter:

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

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

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

Display the emp records who joined in 1981 4th quarter:

```
SELECT ename, hiredate
FROM emp
WHERE to_char(hiredate,'yyyy')=1981 AND
To_char(hiredate,'Q') = 4;
```

Display the emp records who joined on SUNDAY:

ENAME	HIREDATE

```
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


SELECT ename, hiredate
FROM emp
WHERE RTRIM(to_char(hiredate,'DAY'))='SUNDAY';
          RTRIM('SUNDAY3spaces')
                    SUNDAY = SUNDAY      TRUE


Display all emp names and hiredates.
Display horedates in IND date format [DD/MM/YYYY]:
```

ENAME	HIREDATE

```
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 symbol, currency name, thousand separator, decimal point and decimal places we need to convert NUMBER to CHAR [string].

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)
, (or) G	Thousand separator (,)
. (or) D	Decimal point (.)
9	Digit (0 to 9)

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, and
2 decimal places:

ENAME	SAL

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

NOTE:

To see NLS parameters:

Login as DBA:
username: system

```
SQL> SHOW PARAMETERS 'NLS'
```

Output:

NLS PARAMETERS	DEFAULT VALUE
NLS_CURRENCY	$
NLS_ISO_CURRENCY	AMERICA

Display 5000 as ¥5000.00:

```
SELECT to_char(5000, 'L9999.99', 'nls_currency=¥')
FROM dual;
```

Output:
¥5000.00

Display 5000 as JPY5,000.00:

```
SELECT to_char(5000, 'C9,999.99', 'nls_iso_currency=JAPAN')
FROM dual;
```

To_date():

- **It can be used to convert STRING to DATE.**
- **To insert DATE values, we use it.**

Syntax:

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

Example:

To_date('25-DEC-2024')	25-DEC-24
To_date('25/12/2024')	ERROR
To_date('25/12/2024', 'DD/MM/YYYY')	25-DEC-24
To_date('12/25/2024', 'MM/DD/YYYY')	25-DEC-24

to_date('DECEMBER 25 2024','MONTH DD YYYY')	25-DEC-24

Example:

```
T1
  F1 DATE
---------------
25-DEC-24
15-AUG-23
17-FEB-25
```

```
CREATE TABLE t1(f1 DATE);

INSERT INTO t1 VALUES('25-DEC-2024');
Output:
F1
---------
25-DEC-24
```

string → date : Implicit conversion

```
INSERT INTO t1 VALUES(to_date('15-AUG-2023'));
Output:
F1
-------
15-AUG-23
```

string → date : Explicit conversion

```
INSERT INTO t1 VALUES(to_date('17/02/2025'));
Output:
ERROR

INSERT INTO t1 VALUES(to_date('17/02/2025', 'DD/MM/YYYY'));
```

To_number():

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

Syntax:

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

Examples:

To_number('123')	123
To_number('123.45')	123.45

To_number('$5000.00')	ERROR
To_number('$5000.00', 'L9999.99')	5000
To_number('USD5,000.00', 'C9,999.99')	5000

Aggregate Functions / Group Functions / Multi row functions:

T1

F1
10
20
30

Sum(f1)	60
Max(f1)	30
Min(f1)	10
Avg(f1)	20
Count(f1)	3

ORACLE SQL provides following aggregate functions:

- **Sum()**
- **Max()**
- **Min()**
- **Avg()**
- **Count()**

Sum()	**It is used to find sum of a set of values.** **Syntax:** **Sum(<column>)**
Max()	**It is used to find max value in a set of values.** **Syntax:** **Max(<column>)**
Min()	**It is used to find min value in a set of values.**

	Syntax: Min(<column>)
Avg()	It is used to find avrg value in a set of values. Syntax: Avg(<column>)
Count()	It is used to find number of records or number of column values. Syntax: Count(* / <column>)

Examples:

Find sum of salaries of all emps:

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

Find max sal and min sal in all emps:

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

Find avrg salary on all emps:

```
SELECT avg(Sal) FROM emp;
```

Find number of records in emp table:

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

Find sum of salaries of all managers:

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

Find sum of salaries of deptno 30:

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

Find max sal and min sal in deptno 30:

```
SELECT max(sal), min(sal)
FROM emp
WHERE deptno=30;
```

Find max sal and min sal in all clerks:

```
SELECT max(sal), min(sal)
FROM emp
WHERE job='CLERK';
```

Find number of managers:

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

Find number of emps in deptno 30:

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

Find avrg salary on all managers:

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

Find how many emps are getting comm:

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

NOTE:

```
SELECT count(*), count(8), count(100) FROM emp;
```

Output:

COUNT(*)	COUNT(8)	COUNT(100)
15	15	15

Count(*) gives number of rows

Count(<any_number> gives number of rows

Difference b/w count(*), count(<any_number>):

count(*)	Count(8)
• It counts records • It is slower	• It counts 8s • It is faster

Date Functions:

Sysdate
Systimestamp
Add_months()
Months_between()
Last_day()
Next_day()

sysdate	**It returns current system date**
systimestamp	**It returns current system date and time**

Display today's date:

```
SELECT sysdate FROM dual;
```

Display toda's date and current time:

```
SELECT systimestamp 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 24 hrs format:

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

Display date only from systimestamp:

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

Add_Months():

- **It is used to add/subtract months to/from specific date.**

Syntax:

```
Add_months(<date>, <number_of_months>)
```

Examples:

Add 2 days to today's date:

```
SELECT sysdate+2 FROM dual;
```

Add 2 months to todays' 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;
```

Realtime scenarios:

Orders

Orderid	Pid	Cid	Ordered_date	Delivery_date
123456	..	..	sysdate	sysdate+4

Products

PID	PNAME	Mfrd_date	Expiry_date
1234	X	sysdate	Add_months(sysdate, 3)

Employee

EMPID	ENAME	Date_Of_Birth	Date_Of_Retirement
1001	A	25-DEC-2000	Add_months(date_of_birth, 60*12)

CMs_List

STATE	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(1001,'A',sysdate);

INSERT INTO emp(empno,ename,hiredate)
```

```
VALUES(1002,'B',sysdate-1);

INSERT INTO emp(empno,ename,hiredate)
VALUES(1003,'C',add_months(sysdate,-1));

INSERT INTO emp(empno,ename,hiredate)
VALUES(1004,'D',add_months(sysdate,-12));

COMMIT;
```

Display the emp records who joined today:

ENAME	HIREDATE

```
SELECT ename, hiredate
FROM emp
WHERE trunc(hiredate) = trunc(sysdate);
```

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-20	40000
2-JAN-20	40500
..	..
.	
2-AUG-25	100000

Find today's gold rate?

Find yesterday's gold rate?

Find 1 month ago gold rate?

Find 1 year ago gold rate?

Assignment:

SALES

DATEID	AMOUNT
1-JAN-22	150000
2-JAN-22	110000
..	
2-AUG-25	125000

Find today's sales?
yesterday's
1 month ago
1 year ago

Months_Between():

- **It is used to find difference between 2 date values in the form of months.**
- **It returns number of months.**

Syntax:

Months_Between(<date1>, <date2>)

Examples:

Months_Between('4-AUG-2025', '4-JUL-2025')	1
Months_Between('4-AUG-2025', '4-AUG-2024')	12

Months_Between('4-AUG-2025', '4-AUG-2024')/12	12/12 = 1

Examples:

Display all emp records along with their experience:

ENAME	HIREDATE	EXP

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

Display all emp records along with their experience.
Display experience in the form of years and months:

ENAME	HIREDATE	YEARS	MONTHS

15 months	TRUNC(15/12) = 1 MOD(15, 12) = 3	1 year 3 months
30 months	TRUNC(30/12) = 2 MOD(30, 12) = 6	2 years 6 motnhs

```
12)15(1
   12
-----------
   3
```

```
12)30(2
   24
-----------
   6
```

ENAME	HIREDATE	YEARS	MONTHS

```
SELECT ename, hiredate,
TRUNC(Months_between(sysdate, hiredate)/12) AS years,
MOD(TRUNC(Months_Between(sysdate, hiredate)), 12) AS months
FROM emp;
```

Assignment:
Calculate your age in the form of years and months:

```
Months_between(sysdate, DOB)
```

Last_day():

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

Syntax:

```
Lat_day(<date>)
```

Examples:

Last_day(sysdate)	31-AUG-25
Last_day('17-FEB-2025')	28-FEB-25
Last_day('17-FEB-2024')	29-FEB-24

Find next month first date:

```
SELECT last_day(sysdate)+1 FROM dual;
```

Current month first date:

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

Last_day('4-JUL-25')
31-Jul-25 + 1
1-AUG-25

Next_day():

- It is used to find next date based on weekday.
- For example, find next Friday date
  find next Sunday date

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

Examples:

Find next Monday date:

```
SELECT next_day(sysdate, 'mon') FROM dual;
```

Find next Sunday date:

```
SELECT next_Day(sysdate, 'sun') 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()

Syntax of ORDER BY:
ORDER BY <column> ASC/DESC

Example:
Total marks: 1000

ORDER BY marks DESC

MARKS
500

MARKS	RANK	DENSE_RANK
950	1	1
950	1	1

gap

no gaps

```
----------
500
780
950
840
400
950
840
600
780
450
780
```

MARKS	RANK	DENSE_RANK
950	1	1
950	1	1
840	3	2
840	3	2
780	5	3
780	5	3
780	5	3
600	8	4
500	9	5
450	10	6
400	11	7

no gaps

RANK():

- **It is used to apply ranks to records according to specific column order.**
- **Gaps may be there in ranking if multiple values are same.**

Syntax:

```
RANK() OVER(ORDER BY <column> ASC/DESC)
```

DENSE_RANK():

- **It is used to apply ranks to records according to specific column order.**
- **No Gaps will be there in ranking even if multiple values are same.**

Syntax:

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

Examples:

Display all emp names and salaries.
Apply ranks them according to salary descending order:

ENAME	SAL	RANK

```
SELECT ename, sal,
Rank() OVER([PARTITION BY <column>] ORDER BY sal DESC) AS rank
FROM emp;
```

(or)

```
SELECT ename, sal,
Dense_Rank() OVER([PARTITION BY <column>] ORDER BY sal DESC) AS rank
FROM emp;
```

Display all emp names and hiredates.
Apply ranks them according to seniority:

ENAME	HIREDATE	RANK

```
SELECT ename, hiredate,
Dense_rank() OVER(ORDER BY hiredate ASC) AS rank
FROM emp;
```

Display all emp records.
Apply ranks to them according to salary descending order.
If salary is same apply rank according to seniority:

ENAME	SAL	HIREDATE	RANK

```
SELECT ename, sal, hiredate,
Dense_rank() OVER(ORDER BY sal DESC hiredate ASC) AS rank
FROM emp;
```

NOTE:

ORDER BY	**Used to arrange the records in ASC or DESC order**
PARTITION BY	**Used to group the records according to specific column**

Example:

Display all emp records.
Apply ranks to them according to salary descending order with in dept:

ENAME	DEPTNO	SAL	RANK

```
BREAK ON deptno SKIP 1 DUPLICATES

SELECT ename, deptno, sal,
Dense_rank() OVER(PARTITION BY deptno ORDER BY sal DESC) AS rank
FROM emp;
```

Display all emp records.
Apply ranks to them according to salary descending order with in job:

PARTITION BY job ORDER BY sal DESC

CLERK	10000
MANAGER	40000
CLERK	25000
MANAGER	20000

CLERK	10000
CLERK	25000
CLERK	18000

CLERK	25000
CLERK	18000
CLERK	10000

MANAGER	40000
CLERK	25000
MANAGER	20000
MANAGER	30000
CLERK	18000

CLERK	18000

MANAGER	40000
MANAGER	20000
MANAGER	30000

CLERK	10000

MANAGER	40000
MANAGER	30000
MANAGER	20000

Display all emp records.
Apply ranks to them according to salary descending order with in job:

ENAME	JOB	SAL	RANK

```
BREAK ON job SKIP 1 DUPLICATES

SELECT ename, job, sal,
Dense_rank() OVER(PARTITION BY job ORDER BY sal DESC) AS rank
FROM emp;

CLERA BREAKS
```

Example:

Display all emp records.
Apply ranks to them with in dept according to seniority:

ENAME	DEPTNO	HIREDATE	RANK

```
BREAK ON deptno SKIP 1 DUPLICATES

SELECT ename, deptno, hiredate,
Dense_rank() OVER(PARTITION BY deptno ORDER BY hiredate ASC) AS rank
FROM emp;
```

Row_Number():

- **It is used to apply row numbers to records according to specific column order.**

Syntax:

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

Examples:
Display all emp records.
Apply row numbers to them according to emp names alphabetical order:

SNO	ENAME	SAL

```
SELECT row_number() OVER(ORDER BY ename ASC) AS sno,
Ename, sal
FROM emp;
```

Display all emp records.
Apply row numebrs to them according to empnos ascending order:

SNO	EMPNO	ENAME	SAL

```
SELECT row_number() OVER(ORDER BY empno ASC) AS sno,
Empno, ename, sal
FROM emp;
```

Display all emp records.
Apply row numbers to them with in dept according to empno ascending order:

SNO	EMPNO	DEPTNO	SAL

```
BREAK ON deptno SKIP 1 DUPLICATES

SELECT row_number() OVER(PARTITION BY deptno
ORDER BY empno ASC) AS sno,
Empno, deptno, sal
FROM emp;
```

Number Functions:

Sqrt()	Mod()
Power()	
Sign()	Ceil()
Abs()	Floor()
	Trunc()
	Round()

Sqrt():

- It is used to find square root value.

Syntax:

```
Sqrt(<number>)
```

Examples:

Sqrt(100)	10
Sqrt(25)	5

Power():

- **It is used to find power value.**

Syntax:

Power(<number>, <power>)

Examples:

Power(2,3)	**8**
Power(5,2)	**25**

Sign():

- **It is used to check sign of the number.**
- **If num is +ve, it returns 1**

-ve -1

0 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 -ve.**

Syntax:

Abs(<number>)

Examples:

Abs(25)	**25**
Abs(-25)	**25**

Mod():

- **It is used to find remainder value.**

Syntax:

Mod(<number>, <divisor>)

Examples:

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

Mod(10,4)	2

Ceil() and Floor():

Ceil():

- It is used to get round up value.

Syntax:

Ceil(<number>)

Floor():

- It is used to get round down value.

Syntax:

Floor(<number>)

678.5678

Examples:

Ceil(678.5678)	679
Floor(678.5678)	678

Trunc():

- It is used to remove decimal places.

Syntax:

Trunc(<number> [, <no of decimal places>])

Examples:

Trunc(123.45678)	123
Trunc(123.45678, 1)	123.4
Trunc(123.45678, 2)	123.45
Trunc(123.45678, 3)	123.456

2nd arg can be -ve.

If 2nd arg is -ve, it will not give decimal places.

-1	Rounds in 10s	10,20,30,....
-2	Rounds in 100s	100, 200, 300, ...
-3	Rounds in 1000s	
.		
.		

Trunc(456.678, -1)	450 and 460 o/p: 450

Trunc(567.789, -1)	560 and 570 o/p: 560
Trunc(678.4567, -2)	600 and 700 o/p: 600
Trunc(4567.58923, -3)	4000 and 5000 o/p: 4000

Round():

- **It is same as trunc(). But, it considers avrg.**
- **If value is avrg or above avrg, it returns upper value.**
- **If value is below avrg, it returns lower value.**

Syntax:

Round(<number>, <no of decimal places>)

Examples:

Trunc(123.6789)	123 and 124 o/p: 123
Round(123.6789)	123 and 124 Avrg: 123.5 o/p: 124

Avrg of 123 and 124

123+124 = 247
247/2 = 123.5

Trunc(123.45678)	123 and 124 123
Round(123.45678)	123 and 124 Avrg: 123.5 123

TRUNC(123.56789, 1)	123.5
ROUND(123.56789, 1)	123.6

ROUND(123.46789, 1)	123.5
ROUND(123.43789, 1)	123.4

TRUNC(123.46789, 2)	123.46
ROUND(123.46789, 2)	123.47
ROUND(123.46389, 2)	123.46

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

Round() considers avrg
Trunc() never considers avrg

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(157.4567, -1)	150 and 160 o/p:150
ROUND(157.4567, -1)	150 and 160 avrg: 155 160

150+160 = 310
310/2 = 155

TRUNC(153.4567, -1)	150 and 160 o/p: 150
ROUND(153.4567, -1)	150 and 160 Avrg: 155 o/p: 150

TRUNC(567.678, -2)	500 and 600 o/p: 500
ROUND(567.678, -2)	500 and 600 Avrg: 550 600

Other Functions:

NVL()
NVL2()

Decode()

USER
UID

NVL():

- **It is used to replace nulls with some other value.**

Syntax:

NVL(<arg1>, <arg2>)

If arg1 is not null, it returns arg1
If arg1 is null, it returns arg2

Examples:

NVL(100, 200)	100
NVL(null, 200)	200

Examples:

Calculate total salary of each empolyee:

ENAME	SAL	COMM
A	10000	2000
B	25000	5000
C	18000	

SAL+NVL(COMM, 0)

10000+nvl(2000, 0) => 10000+2000 = 12000
25000+nvl(5000, 0) => 25000+5000 = 30000
18000+nvl(null, 0) => 18000+0 = 18000

ENAME	SAL	COMM	TOTAL SALARY

```
SELECT ename, sal, comm,
Sal+NVL(comm,0) AS "TOTAL SALARY"
FROM emp;
```

Display all emp records along with commissions.
If comm is NULL, display it as N/A [Not Applicable]:

ENAME	SAL	COMM

```
SELECT ename, sal,
Nvl(to_char(comm), 'N/A') AS comm
FROM emp;
```

Assignment:
STUDENT

SID	SNAME	M1 NUMBER(3)
1001	A	70
1002	B	0
1003	C	60
1004	D	
1005	E	89
1006	F	

Display nulls with AB:

NVL(m1, 'AB')

NVL2():

- **It is used to replace nulls and not nulls.**

Syntax:

NVL2(<arg1>, <arg2>, <arg3>)

If arg1 is not null, it returns arg2
If arg1 is null, it returns arg3

Examples:

NVL2(100, 200, 300)	**200**
NVL2(null, 200, 300)	**300**

Diff b/w NVL() and NVL2():

NVL	NVL2()
• It is used to replace nulls • It takes 2 arguments	• It is used to replace nulls and nulls • It takes 3 arguments

Example:

Modify commissions as following:
If emp is getting commission then increase 1000 rupees.
If emp is not getting commission then set it as 900.

```
UPDATE emp
SET comm= NVL2(comm, comm+1000, 900);
```

Decode():

- **It is used to implement switch control structure in SQL.**

Syntax:

```
Decode(<column>,
        <value1>, <extression1>,
        <value2>, <expression2>,
        .
        .
        <else expression>)
```

Example:

Display all emp records along with job titles.

If job is PRESIDENT then display it as BIG BOSS
MANAGER BOSS
OTHERS SUBORDINATE.

ENAME	JOB	SAL

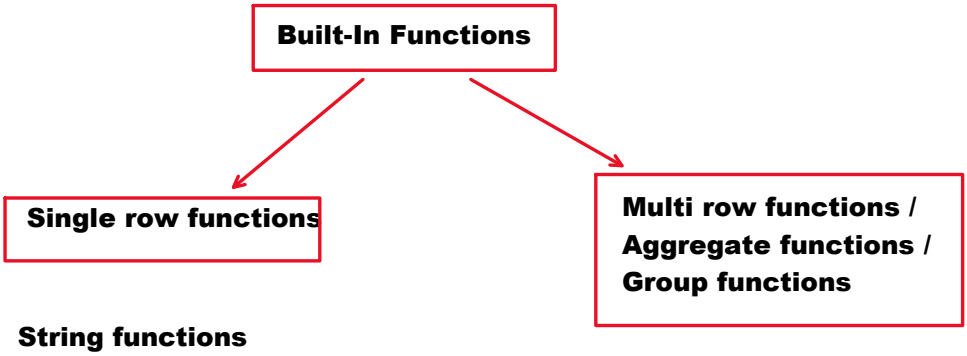
```
SELECT ename,
Decode(JOB,
        'PRESIDENT', 'BIG BOSS',
        'MANAGER', 'BOSS',
        'SUBORDINATE')  AS job,
Sal
FROM emp;
```

User:
- It returns current user name.

UID:
- It returns current user id.

Example:
Display current user id and name:

```
SELECT uid, uname FROM dual;
```



String functions
Conversion functions
Analytic
.
. etc

Single row function:
If 1 function call is applied on 1 row then it is called "single row function".

Multi row function:

If 1 function call is applied on multiple rows then it is called "Multi row function".

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

Lower() => single row function

SAL **Max(Sal) => multi row function**

10000

25000

18000

Summary:

String functions	**Lower() initcap() Upper()** **Lpad() Rpad()** **Ltrim() Rtrim() Trim()** **Substr() Instr()** **Length() Concat()** **Replace() tanslate()**
Conversion	**To_char()** **To_date()** **To_number()**
Aggregate	**Sum() max() min()** **Count() avg()**
Date	**Add_months()** **Months_between()** **Sysdate** **Systimestamp** **Last_day()** **Next_day()**
Analytic	**Dense_rank() Rank()** **Row_number()**
Number	**Trunc() Round()** **Ceil() Floor()** **Mod() Power()** **Sqrt() abs()** **Sign()**
Other	**NVL() NVL2()** **User Uid** **Decode()**

CLAUSES

Tuesday, August 5, 2025 10:48 AM

CLAUSES of SELECT command:

SQL	ENGLISH
QUERIES	SENTENCES
CLAUSES	WORDS

- CLAUSE is a part of query.
- Every clause has specific purpose.
- Every query is made up of with clauses.

Syntax of SELECT command:

```
SELECT ALL/DISTINCT <column_list>
[FROM <table_list>]
[WHERE <condition>]
[GROUP BY <grouping_column_list>]
[HAVING <group_condition>]
[ORDER BY <column> ASC/DESC, <column> ASC/DESC, ...]
[OFFSET <number> ROW/ROWS]
[FETCH FIRST/NEXT <number> ROW/ROWS ONLY];
```

In oracle 21c,	FROM is mandatory SELECT 100+200; Output: ERROR: FROM not found SELECT sysdate; Output: ERROR
In oracle 23ai,	FROM is optional

	SELECT 100+200; Output: 300 SELECT sysdate; Output: 07-AUG-25

Display all managers names and their salaries:

ENAME	SAL

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

SELECT	• It is used to specify column list. Syntax: SELECT <column_list> Example: SELECT ename, sal
FROM	• It is used to specify table list. Syntax: FROM <table_list> Examples: FROM emp FROM emp, dept
WHERE	• It is used to specify filter condition. • WHERE condition will be applied on row. • It filters the rows

	Syntax: **WHERE <condition>** **Examples:** **WHERE job='MANAGER'** **WHERE deptno IN(10,30)**

WHERE job='MANAGER'

JOB	WHERE job='MANAGER'	
---	.....	
CLERK	CLERK=MANAGER	F
ANALYST	ANALYST=MANAGER	F
MANAGER	MANAGER=MANAGER	T

ORDER BY:

- **It is used to arrange the records in ascending or descending order.**
- **Default order is: ASC.**

Syntax:
ORDER BY <column> ASC/DESC, <column> ASC/DESC, ..

Examples on ORDER BY:

Display all emp names and salaries.
Arrange emp names in alphabetical order:

ENAME	SAL

```
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;
```

ENAME	1
SAL	2

(or)

```
SELECT *
FROM emp
ORDER BY 2;
```

*	Empno, ename, job, mgr, hiredate, sal, comm, deptno, mail_id 2

Display all emp names and salaries.
Arrange salaries in descending order:

ENAME	SAL

```
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 according to seniority:

ENAME	HIREDATE

```
SELECT ename, hiredate
FROM emp
ORDER BY hiredate ASC;
```

Display all emp records.
Arrange them in ascending order according to deptno:

ENAME	DEPTNO	SAL

```
BREAK ON deptno SKIP 1 DUPLICATES

SELECT ename, deptno, sal
FROM emp
ORDER BY deptno ASC;
```

Arranging records in order according to multiple columns:

Display all emp records.
Arrange them in ascending order according to deptno.
With in dept arrange salaries in descending order:

ENAME	DEPTNO	SAL

```
SELECT ename, deptno, sal
FROM emp
ORDER BY deptno ASC, sal DESC;
```

CASE-1: deptnos are different

```
DEPTNO
----------
20
10
```

```
10
20
```

CASE-2: deptnos are same

DEPTNO	SAL
10	15000
10	20000

10	20000
10	15000

Display all emp records.
Arrage them in ascending order according to deptno.
With in dept arrange salaries in descending order.
If salary is same arrange them according to seniority:

ENAME	DEPTNO	SAL	HIREDATE

```
SELECT ename, deptno, sal, hiredate
FROM emp
ORDER BY deptno ASC, sal DESC, hiredate ASC;
```

NOTE:

- In ASC, NULLs will be displayed last.
- In DESC, NULLS will be displayed first.

SAL
5000
8000
Null
7000
6000

ORDER BY sal ASC

SAL
5000
6000
7000
8000
NULL

ORDER BY sal DESC

SAL
NULL
8000
7000
6000
5000

Examples:

Display all emp names and salaries.
Arrange salaries in descending order.
Display nulls last:

ENAME	SAL

```
SELECT ename, sal
FROM emp
ORDER BY sal DESC NULLS LAST;
```

Display all emp names and salaries.
Arrange salaries in ascending order.
Display nulls first:

ENAME	SAL

```
SELECT ename, sal
FROM emp
ORDER BY sal ASC NULLS FIRST;
```

GROUP BY:

- GROUP BY is used to group the records according to specific column(s).
- On these groups we can apply aggregate function.
- It is used for data analysis.
- It gives summarized data from detailed data.

Syntax:

```
GROUP BY <column_list>
```

Examples on GROUP BY:

Find dept wise sum of salaries:

DEPTNO	SUM_OF_SAL
10	?
20	?

```
SELECT deptno, sum(Sal) AS sum_of_sal
FROM emp
GROUP BY deptno
ORDER BY 1;
```

Execution order [oracle 21c]:

FROM
WHERE
GROUP BY
HAVING
SELECT
DISTINCT
ORDER BY
OFFSET
FETCH

Example:

FROM emp:
Selects entire table

EMP

EMPNO	ENAME	DEPTNO	SAL
1001	A	30	20000
1002	B	30	10000
1002	C	10	30000
1003	D	10	20000
1004	E	20	50000
1005	F	20	10000

GROUP BY deptno:
It groups the records according to deptno

1001	A	30	20000
1002	B	30	10000

Sum(Sal) => 30000

1002	C	10	30000
1003	D	10	20000

Sum(sal) => 50000

1004	E	20	50000

Sum(Sal) => 60000

1004	E	20	30000
1005	F	20	10000

Sum(Sal) => 60000

SELECT deptno, sum(Sal) AS sum_of_sal:

DEPTNO	SUM_OF_SAL
30	30000
10	50000
20	60000

ORDER BY 1:

Arranges deptnos in ascending order.

DEPTNO	SUM_OF_SAL
10	50000
20	60000
30	30000

Example:

Dept wise max sal and min sal:

DEPTNO	MAX_SAL	MIN_SAL

```
SELECT deptno, max(Sal) AS max_sal, min(Sal) AS min_sal
FROM emp
GROUP BY deptno
ORDER BY 1;
```

Find dept wise no of emps:

DEPTNO	NO_OF_EMPS
10	?
20	?

SELECT deptno, count(*) AS no_of_emps

```
FROM emp
GROUP BY deptno
ORDER BY 1;
```

Find dept wise avrg sal:

```
SELECT deptno, avg(Sal) AS avrg_Sal
FROM emp
GROUP BY deptno
ORDER BY 1;
```

Assignment:

Find job wise number of emps:

JOB	NO_OF_EMPS
CLERK	?
MANAGER	?

GROUP BY job
Count(*)

Find job wise max sal and min sal:

JOB	MAX_SAL	MIN_SAL
CLERK	?	?
MANAGER	?	?

GROUP BY job
Max(sal)
Min(sal)

Find job wise sum of salaries:

JOB	SUM_OF_SAL
CLERK	?
MANAGER	?

GROUP BY job
Sum(sal)

Examples:

Find year wise number of emps joined in organization:

YEAR	NO_OF_EMPS
1980	?
1981	?

```
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 number of emps joined in organization:

Q	NO_OF_EMPS
1	?
2	?

```
SELECT to_char(hiredate,'Q') AS Q,
Count(*) AS no_of_emps
FROM emp
GROUP BY to_char(hiredate,'Q')
ORDER BY 1;
```

Assignment:

Find month wise number of emps:

MONTH	NO_OF_EMPS
JAN	?
FEB	?

Find weekday wise number of emps:

MONTH	NO_OF_EMPS
SUN	?
MON	?

Assignment:

PERSON

PID	PNAME	STATE	GENDER	AADHAR
		TG	M	
		TG	F	
		AP	M	

		AP	F	

Find state wise population. => GROUP BY state

Find gender wise population. => GROUP BY gender

Assignment:

SALES

DATEID	AMOUNT
1-JAN-2023	120000
2-JAN-2023	100000
..	
8--AUG-2025	130000

Year wise sales

Quarter wise sales

Grouping Records according to multiple columns:

DEPTNO	JOB
20	ANALYST
20	ANALYST
20	CLERK
20	CLERK
20	MANAGER
30	SALESMAN
30	SALESMAN
30	CLERK
30	CLERK
30	MANAGER

Dept wise no of emps
GROUP BY deptno => 2 grps

10	?

Job wise no of emps
GROUP BY job => 4 grps

CLERK	?

Dept wise, job wise no of emp

GROUP BY deptno, job => 6 grps

20	ANALYST	
20	CLERK	
20	MANAGER	

30	SALESMAN	
30	CLERK	
30	MANAGER	

Examples:

Find dept wise, with in dept job wise no of emps:

DEPTNO	JOB	NO_OF_EMPS
20	CLERK	?
20	MANAGER	?
30	CLERK	?
30	MANAGER	?

```
SELECT deptno, job, count(*) AS no_of_emps
FROM emp
GROUP BY deptno, job
ORDER BY 1;
```

Find dept wise, with in dept job wise sum of sal:

DEPTNO	JOB	SUM_OF_SAL
20	CLERK	?
20	MANAGER	?
30	CLERK	?
30	MANAGER	?

```
SELECT deptno, job, sum(Sal) AS sum_of_sal
FROM emp
GROUP BY deptno, job
ORDER BY 1;
```

Example:

PERSON

PID	PNAME	STATE	GENDER	AADHAR
...	..	TG	M	..

Find state wise population
GROUP BY state

PID	PNAME	STATE	GENDER	AADHAR
...	..	TG	M	..
		TG	M	
		TG	F	
		TG	F	
		AP	M	
		AP	M	
		AP	F	
		AP	F	

Find state wise population
GROUP BY state

Find gender wise population
GROUP BY gender

Find state wise, with in state gender wise population
GROUP BY state, gender
Output:

STATE	GENDER	NO_OF_PEOPLE
TG	M	?
TG	F	?
AP	M	?
AP	F	?

```
SELECT state, gender, count(*) AS no_of_people
FROM person
GROUP BY state, gender
ORDER BY 1;
```

Example:
Find year wise, with in year quarter wise no of emps joined in organization:

YEAR	Q	NO_OF_EMPS
1980	1	?
	2	?
	3	?
	4	?
1981	1	?
	2	?
	3	?

	4	?

```
BREAK ON year SKIP 1

SELECT to_char(hiredate,'yyyy') AS year,
To_char(hiredate,'Q') AS Q,
Count(*) AS no_of_emps
FROM emp
GROUP BY to_char(hiredate,'yyyy'), To_char(hiredate,'Q')
ORDER BY 1;
```

Rollup() and Cube():

Rollup():

- It is used to calculate sub totals and grand total.
- We call it from GROUP BY clause and we pass grouping column list as arguments.
- It calculates sub totals according to first column in grouping column list.

Syntax:

```
GROUP BY Rollup(<grouping_column_list>)
```

Example:

```
GROUP BY Rollup(deptno, job)
```

Cube():

- It is used to calculate sub totals and grand total.
- We call it from GROUP BY clause and we pass grouping column list as arguments.
- It calculates sub totals according to all columns in grouping column list.

Syntax:

```
GROUP BY Cube(<grouping_column_list>)
```

Example:

GROUP BY Cube(deptno, job)

Example:
Find dept wise, with in dept wise job wise no of emps.
Find sub totals accrding to deptno [rollup()]:

DEPTNO	JOB	NO_OF_EMPS
20	CLERK	?
	MANAGER	?
	20 SUB TOTAL	?
30	CLERK	?
	MANAGER	?
	30 SUB TOTAL	?
	GRAND TOTAL	?

```
BREAK ON deptno SKIP 1 DUPLICATES

SELECT deptno, job, count(*) AS no_of_emps
FROM emp
GROUP BY Rollup(deptno, job)
ORDER BY 1;
```

Find dept wise, with in dept wise job wise no of emps.
Find dept wise subtotal and job wise subtotal [Cube]:

DEPTNO	JOB	NO_OF_EMPS
20	CLERK	?
	MANAGER	?
	20 SUB TOTAL	?
30	CLERK	?
	MANAGER	?
	30 SUB TOTAL	?
	CLERK SUB TOTAL	?
	MANAGER SUB TOTAL	?
	GRAND TOTAL	?

```
BREAK ON deptno SKIP 1 DUPLICATES

SELECT deptno, job, count(*) AS no_of_emps
FROM emp
GROUP BY Cube(deptno, job)
ORDER BY 1;
```

Having:

- It is used to write conditions on groups.
- It filters the groups.
- It is applied on result of GROUP BY.
- It cannot be used without GROUP BY.

Syntax:

```
GROUP BY <grouping_col_list>
HAVING <group_condition>
```

Examples on HAVING:

Display the depts which are spending more than 10000 rupees on their emps:

DEPTNO	SUM(SAL)

```
SELECT deptno, sum(Sal)
FROM emp
GROUP BY deptno
HAVING sum(Sal)>10000;
```

Display the job titles which are having 3 or less emps:

JOB	Count(*)

```
SELECT job, count(*)
FROM emp
GROUP BY job
```

```
HAVING count(*)<=3;
```

Display the deptnos which are having 5 or more emps:

DEPTNo	COUNT(*)

```
SELECT deptno, count(*)
FROM emp
GROUP BY deptno
HAVING count(*)>=5;
```

Differences b/w WHERE and HAVING:

WHERE	HAVING
• WHERE condition is applied on rows.	• HAVING condition is applied 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.
• Aggregate function cannot be used here.	• Aggregate function can be used here.

Execution Order [oracle 21c]:

```
FROM
WHERE
GROUP BY
HAVING
SELECT
DISTINCT
ORDER BY
```

OFFSET
FETCH

OFFSET and FETCH:

- Introduced in ORACLE 12c.
- They improve performance.
- They avoid of writing sub queries.

OFFSET:

- It is used to skip the rows.

Syntax:

```
OFFSET <number> ROW/ROWS
```

FETCH:

- It is used fetch [select] the rows.

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 from emp table:

```
SELECT * FROM emp
OFFSET 5 ROWS
FETCH NEXT 5 ROWS ONLY;
```

Find 2nd max sal:

```
SELECT  DISTINCT sal        ORDER BY sal DESC
```

```
SAL
-------------
5000
7000
4000
7000
6000
3000
7000
```

```
5000
7000
4000
6000
3000
```

```
7000
6000
5000
4000
3000
```

```
OFFSET 1 ROW
FETCH NEXT 1 ROW ONLY
```

```
SELECT DISTINCT sal
FROM emp
ORDER BY sal DESC
OFFSET 1 ROW
FETCH NEXT 1 ROW ONLY;
```

Find 3rd max sal:

```
SELECT DISTINCT sal
FROM emp
ORDER BY sal DESC
OFFSET 2 ROWS
FETCH NEXT 1 ROW ONLY;
```

Find 5th max sal:

```
SELECT DISTINCT sal
FROM emp
ORDER BY sal DESC
OFFSET 4 ROWS
```

```
FETCH NEXT 1 ROW ONLY;
```

Display top 3 salaries:

```
SELECT DISTINCT sal
FROM emp
ORDER BY sal DESC
FETCH FIRST 3 ROWS ONLY;
```

Display top 3 seniors records:

```
SELECT *
FROM emp
ORDER BY hiredate ASC
FETCH FIRST 3 ROWS ONLY;
```

DISTINCT:

- It is used to eliminate duplicate rows.

Syntax:

```
SELECT DISTINCT <column_list>
```

Examples on DISTINCT:

Display the job titles offered by company:

```
SELECT DISTINCT job FROM emp;
```

Display distinct deptnos:

```
SELECT DISTINCT deptno FROM emp
ORDER BY deptno ASC;
```

Display the job titles offered by each dept:

SELECT DISTINCT deptno, job FROM emp;

Summary:

FROM	**Selects entire table**
WHERE	**Filters the rows**
GROUP BY	**Groups the rows**
HAVING	**Filters the groups**
SELECT	**Selects the columns**
DISTINCT	**Eliminates duplicate rows**
ORDER BY	**Arranges records in ASC or DESC**
OFFSET	**Skips the rows**
FETCH	**Fetches the rows**

JOINS

Monday, August 11, 2025 10:32 AM

JOINS:

JOINS concept Goal:
JOINS concept is used to retrieve the data from multiple tables.

College DB

Course
Student
Marks
Fee
Dept
Emp
.
.

JOINS:

- **JOIN is an operation.**
- **In Join operation,**
  one table row will be joined with another table row based on join condition.
- **It is used to retrieve data from multiple tables.**
- **Join condition decides which record in a table should be joined with which record in another table.**

Types of Joins:

4 types:

- **Inner Join**
  - **Equi Join**
  - **Non-Equi Join**
- **Outer Join**
  - **Left Outer Join**
  - **Right Outer Join**
  - **Full Outer Join**
- **Self-Join / Recursive Join**
- **Cross Join / Cartesian join**

Inner Join:

- **Inner join = matched records only**
- **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:

s.sid = m.sid

STUDENT s

SID	SNAME	SCITY
1001	A	HYD
1002	B	MUM
1003	C	BLR

MARKS m

SID	MATHS	PHY	CHE
1001	70	90	80
1002	65	82	73

Display student details along with maths subject marks:

SID	SNAME	MATHS

STUDENT MARKS

```
SELECT sid, sname, maths
FROM student, marks
WHERE sid = sid;
Output:
ERROR: column ambiguously defined

SELECT sid, sname, maths
FROM student, marks
WHERE student.sid=marks.sid;
Output:
ERROR: column ambiguously defined

SELECT student.sid, sname, maths
FROM student, marks
WHERE student.sid=marks.sid;
```

NOTE:
Above query degrades performance.

```
SELECT student.sid, student.sname, marks.maths
FROM student, marks
WHERE student.sid=marks.sid;
```

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;
```

NOTE:
When we write join query,
Prefix every column name with table name. It improves performance.

To make lengthy table name short use table alias.

Example:

e.deptno=d.deptno

EMP e

EMPNO	ENAME	SAL	DEPTNO
7369	SMITH	800	20
7499	ALLEN	1600	30
7521	WARD	2000	30
7839	KING	5000	10
1001	A	5000	
1002	B	8000	

DEPT d

DEPTNO	DNAME	LOC
10	ACCOUNTING	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

```
SELECT e.ename, e.sal, d.dname, d.loc
FROM emp e, dept d
WHERE e.deptno=d.deptno;
```

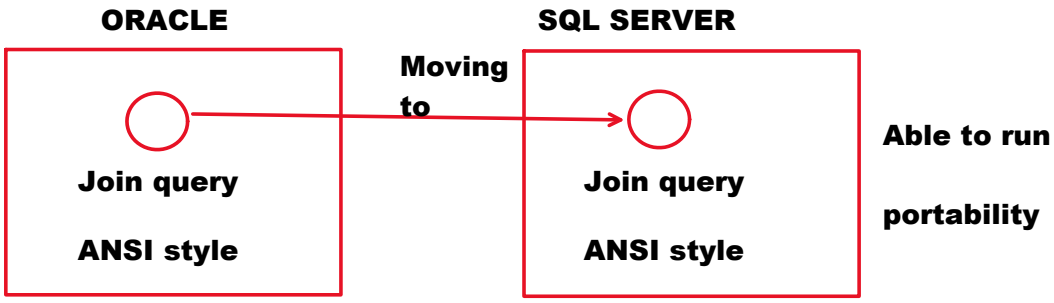
NOTE:
From ORACLE 9i version onwards
We can write join query in 2 styles. They are:

- **ORACLE STYLE / Native style / Non-ANSI style**
- **ANSI STYLE**

ANSI STYLE is best way => portable

ORACLE SQL SERVER

ANSI STYLE is best way => portable



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;
```

NOTE:

In ORACLE style, To separate 2 table names we write , [comma].
In ANSI style, To separate 2 table names we use keyword.

In ORACLE style, we write join condn in WHERE clause.
In ANSI style, we write join condition in ON clause.

Assignment:

STUDENT

SID	SNAME	CID
1001	A	20
1002	B	20
1003	C	10
1004	D	10
1005	E	30

COURSE

CID	CNAME
10	JAVA
20	PYTHON
30	C#.Net

Display snames and cnames.

SNAME	CNAME
A	PYTHON

Student s course c

Display the emp records who are working in SALES dept:

ENAME	DNAME
	SALES

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';
```

EMP

EMPNO	ENAME	SAL	DEPTNO
7369	SMITH	800	20
7499	ALLEN	1600	30
7521	WARD	2000	30
7839	KING	5000	10
1001	A	5000	
1002	B	8000	

DEPT

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

e.deptno =d.deptno

EMPNO	ENAME	SAL	DEPTNO
7369	SMITH	800	20
7499	ALLEN	1600	30
7521	WARD	2000	30
7839	KING	5000	10
1001	A	5000	
1002	B	8000	

DEPTNO	DNAME	LOC
30	SALES	CHICAGO

NOTE:
To see execution plan:

```
SQL> SET AUTOTRACE ON EXPLAIN
```

Display BLAKE record along with dept details:

ENAME	SAL	DNAME	LOC
BLAKE			

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 e.ename='BLAKE';
```

ANSI STYLE:

```
SELECT e.ename, e.sal, d.dname, d.loc
FROM emp e INNER JOIN dept d
ON e.deptno=d.deptno
WHERE e.ename='BLAKE';
```

Equi Join:

- **If join operation is performed based on equality condition then it is called "Equi Join".**

Example:

```
WHERE e.deptno=d.deptno
WHERE s.sid=m.sid
```

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	2500
1002	B	1300
1003	C	1100
1004	D	5000

SALGRADE s

GRADE	LOSAL	HISAL
1	700	1200
2	1201	1400
3	1401	2000
4	2001	3000
5	3001	9999

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;
```

Outer Join:

- **Inner Join = matched records only.**
- **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, Join condition decides left table and right table.

Examples:

WHERE e.deptno = d.deptno

emp e	Left table
dept d	Right table

WHERE d.deptno = e.deptno

dept d	Left table
emp e	Right table

In ANSI STYLE, keyword decides left table and right table.

Examples:

FROM emp e JOIN dept d

emp e	Left table
dept d	Right table

FROM dept d JOIN emp e

dept d	Left table
emp e	Right table

Left Outer Join:

- **Left Outer Join = matched + unmatched from left table.**
- **Left Outer Join gives matched records and unmatched records from left table.**
- **In ORACLE STYLE, we use outer join operator.**
  Outer join operator symbol: (+).
  For left outer join write (+) symbol at right side.
- **In ANSI STYLE, we use keyword: LEFT [OUTER] JOIN.**

Example on Left Outer Join:

Display emp details along with dept details.

Also display the emps to whom dept is not assigned.

ENAME	SAL	DNAME	LOC
SMITH	800	RESEARCH	DALLAS
A	5000		

Matched
Unmatched from emp [left table]

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 gives matched records and unmatched records from right table.
- In ORACLE STYLE, write (+) symbol at left side.
- In ANSI STYLE, use the keyword: RIGTH [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
SMITH	800	RESEARCH	DALLAS
		OPERATIONS	BOSTON

Matched
Unmatched from dept (right)

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 + um from left + um from right**
- **Full outer join gives matched records unmatched from left and right tables.**
- **In ORACLE STYLE,**
  We write UNION operator between Left join and right join.

Left outer join	**= matched + unmatched from left**
UNION	
Right outer join	**= matched + unmatched from right**

Full outer join = matched + um from left + um from right

- **In ANSI STYLE, use the keyword: FULL OUTER JOIN.**

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 which are not having emps:

ENAME	SAL	DNAME	LOC
SMITH	800	RESEARCH	DALLAS
A	5000		
		OPERATIONS	BOSTON

Matched
Um from emp (left)
Um from dept (right)

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;

ASNI 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 = unmatched from left table**
- **Right Outer Join + condn = unmatched from right table**
- **Full Outer Join + condns = um from left and right tables**

Left Outer Join + condition:

Display the emps to whom dept is not assigned:

ENAME	DNAME
A	

Unmatched from emp (emp => left table)

EMP e DEPT d

ORACLE STYLE:

```
SELECT e.ename, d.dname
FROM emp e, dept d
WHERE e.deptno=d.deptno(+) AND d.dname IS null;
```

ANSI STYLE:

```
SELECT e.ename, d.dname
FROM emp e LEFT JOIN dept d
ON e.deptno=d.deptno
WHERE d.dname IS null;
```

Right Outer Join + condn:

Display the depts which are not having emps:

ENAME	DNAME
	OPERATIONS

Unmatched from dept (dept right table)

Display the depts which are not having emps:

ENAME	DNAME
	OPERATIONS

Unmatched from dept (dept right table)

Emp e dept d

ORACLE STYLE:

```
SELECT e.ename, d.dname
FROM emp e, dept d
WHERE e.deptno(+)=d.deptno AND e.ename IS null;
```

ANSI STYLE:

```
SELECT e.ename, d.dname
FROM emp e RIGHT JOIN dept d
ON e.deptno=d.deptno
WHERE e.ename IS null;
```

Full Outer Join + condns:

Display the emps to whom dept is not assigned.
Also display the depts which are not having emps:

ENAME	DNAME
A	
	OPERATIONS

Unmatched from emp (emp => left)
Unmatched from dept (dept = > right)

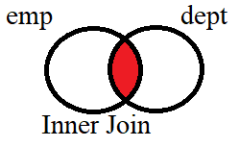
ORACLE STYLE:

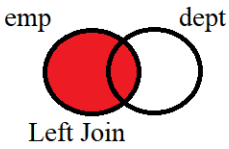
```
SELECT e.ename, d.dname
FROM emp e, dept d
WHERE e.deptno=d.deptno(+) AND d.dname IS null
UNION
SELECT e.ename, d.dname
FROM emp e, dept d
WHERE e.deptno(+)=d.deptno AND e.ename IS null;
```

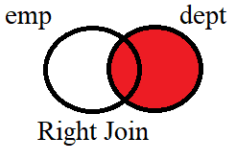
ANSI STYLE:

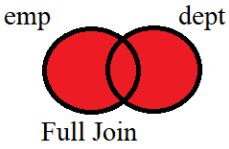
```
SELECT e.ename, d.dname
FROM emp e FULL JOIN dept d
ON e.deptno=d.deptno
WHERE d.dname IS null OR e.ename IS null;
```

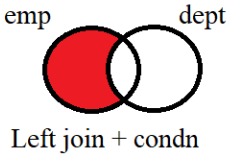
Venn Diagrams of Joins:

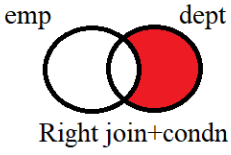


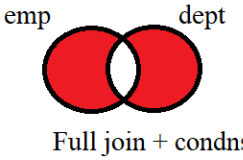












Self-Join / Recursive Join:

- **If a table is joined to itself then it is called "Self-Join / Recursive Join".**
- **In this,**
  One table record will be joined with another record in same table.

Example on Self Join:

Display emp details along with their managers details:

EMP_NAME	EMP_SAL	MGR_NAME	MGR_SAL
B	10000	A	30000

e.mgr=m.empno

EMP e

EMPNO	ENAME	JOB	SAL	MGR
1001	A	MANAGER	30000	
1002	B	CLERK	10000	1001
1003	C	ANALYST	12000	1001
1004	D	MANAGER	25000	
1005	E	CLERK	14000	1004

EMP m

EMPNO	ENAME	JOB	SAL	MGR
1001	A	MANAGER	30000	
1002	B	CLERK	10000	1001
1003	C	ANALYST	12000	1001
1004	D	MANAGER	25000	
1005	E	CLERK	14000	1004

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 emps who are earning more than their managers:

EMP_NAME	EMP_SAL	MGR_NAME	MGR_SAL
	3000		2975

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;
```

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
WHERE e.sal>m.sal;
```

Display the emps who are working under BLAKE:

EMP_NAME	MGR_NAME
	BLAKE

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:

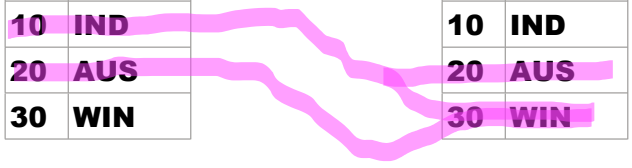
x.cid < y.cid

GROUPA x

CID	CNAME
10	IND
20	AUS
30	WIN

GROUPA y

CID	CNAME
10	IND
20	AUS
30	WIN



Output:
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:

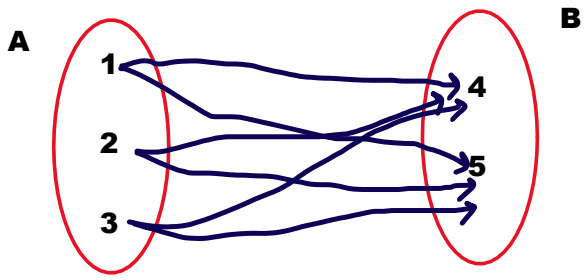
```
SELECT x.cname || ' VS ' || y.cname
FROM groupA x INNER JOIN groupA y
ON x.cid<y.cid;
```

Cross Join / cartesian join:

In maths, cartesian product

A = {1,2,3}
B = {4,5}

A X B ={ (1,4) (1,5)
(2,4) (2,4)
(3,4) (3,5) }



- **In cross join, Each record in one table will be joined with every record in another table.**
- **In this, we will not write join any condition.**

Example:

ICC

GROUPA a

CID	CNAME
10	IND
20	AUS
30	WIN

GROUPB b

CID	CNAME
40	ENG
50	BAN
60	PAK

Output:

IND VS ENG
IND VS BAN
IND VS PAK

AUS VS ENG
AUS VS BAN
AUS VS PAK

WIN VS ENG
WIN VS BAN
WIN VS PAK

Cross join:

ORACLE STYLE:

```
SELECT a.cname || ' VS ' || b.cname
FROM groupA a, group B b;
```

ANSI STYLE:

```
SELECT a.cname || ' VS ' || b.cname
FROM groupA a CROSS JOIN group B b;
```

Assignment:

EMPS e

EMPID	ENAME	PID
1001	A	20
1002	B	20

PROJECTS p

PID	PNAME	DURATION
10	X	2
20	Y	3

EMP_ID	ENAME	PID
1001	A	20
1002	B	20
1003	C	30
1004	D	
1005	E	

PID	PNAME	DURATION
10	X	2
20	Y	3
30	Z	1

ENAME	PNAME

Emps e Projects p

Display emp details along with dept details

ENAME	PNAME
A	Y

Emps e Projects p

Equi join

Display emp details along with dept details.
Also display the emps who are on bench:

ENAME	PNAME
A	Y
D	
E	

Emps e Projects p

Matched
Unmatched from emps (left)

Left outer join

Display emp details along with dept details.
Alos display the projects which are not assigned to any emp

ENAME	PNAME
A	Y
	X

Emps e Projects p

Matched
Unmatched from projects (Right)

Right outer join

Assignment:

Table1 Table2
F1 F1

Table1

F1
1
1
1
2
3
3
3

Table2

F1
1
1
2
2
4
null

Type of join	No of rows
Inner Join (equi)	?
Left outer join	?
Right outer join	?
Full outer join	?
Cross join	?

EMP

EMPNO	ENAME	SAL	DEPTNO
7369	SMITH	800	20
7499	ALLEN	1600	30
7521	WARD	2000	30
7839	KING	5000	10
1001	A	5000	
1002	B	8000	

DEPT

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

SUB QUERIES

Saturday, August 16, 2025 9:52 AM

SUB QUERIES / NESTED QUERIES:

- **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".**

- **Sometimes we may don't know filter condition value. To find it we write sub query.**

- **Sub query must be SELECT only.**
  It cannot be UPDATE / DELETE / INSERT.

- **Main query can be UPDATE or DELETE or INSERT or SELECT.**

- **Sub query must be written in parenthesis.**

Types of sub queries:

2 types:

- **Non-correlated sub query**
  - **Single row sub query**
  - **Multi row sub query**
  - **Inline view / inline sub query**
  - **Scalar sub query**
- **Correlated sub query**

Non-Correlated sub query:

- In this,
  - First, inner query gets executed.
  - Inner query passes value to outer query.
  - Outer query gets executed.

- Inner query gets executed one time.

- It has 4 sub types. They are:
  - Single row sub query
  - Multi row sub query
  - Inline view
  - Scalar sub query

Single row sub query:

- If sub query returns 1 row then it is called "Single row sub query".
- We write it in WHERE clause.
- In WHERE clause, we can write max of 254 sub queries.

Syntax:

```
SELECT <column_list>
FROM <table_name>
WHERE <column> <operator> (<SELECT query>);
```

Examples on single row sub query:

Display the emp records who are earning more than BLAKE:

ENAME	SAL

```
SELECT ename, sal
FROM emp
WHERE sal>(find BLAKE sal);
```

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

```
WHERE sal>(SELECT sal FROM emp WHER ename='BLAKE');
```

Display the emp records whose job title is same as SMITH:

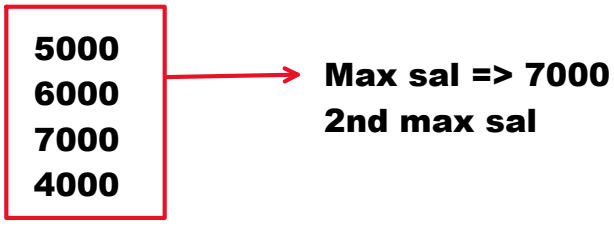
ENAME	JOB	SAL

```
SELECT ename, job, sal
FROM emp
WHERE job=(find SMITH job title);

SELECT ename, job, sal
FROM emp
WHERE job=(SELECT job FROM emp WHERE ename='SMITH');
```

Find 2nd max sal:

```
SAL
---------
5000
8000
6000
7000
4000
```

```
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 3rd max sal:

```
SAL
---------
5000
8000
6000
7000
4000
```

```
SELECT max(sal)
FROM emp
WHERE sal<(find 2nd max sal);
```

```
5000
6000
4000
```


Max sal => 6000
3rd max sal

```
SELECT max(Sal)
FROM emp
WHERE sal<(SELECT max(Sal) FROM emp
WHERE sal<(SELECT max(Sal) FROM emp));
```

```
SAL
---------
5000
8000
6000
7000
4000
```

2nd max sal => 1 SQ
3rd max sal => 2 SQ

10th max sal => 9 SQ

Display the emp record who is earning max sal:

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

Output:
ERROR

NOTE:
When we use GROUP FUNCTION, SELECT clause
Allows either GROUP BY column or GROUP FUNCTION.

```
SELECT *
```

FROM emp
WHERE sal=(find max sal);

SELECT *
FROM emp
WHERE sal = (SELECT max(Sal) FROM emp);

Assignment:

- Display the emp record who is earning 2nd max sal
- Display the emp record who is earning 3rd max sal
- Display the emp record who is earning min sal

Find most senior record:

SELECT *
FROM emp
WHERE hiredate=(find most senior hiredate);

SELECT *
FROM emp
WHERE hiredate = (SELECT min(hiredate) FROM emp);

Find most junior record:

SELECT *
FROM emp
WHERE hiredate=(find most junior hiredate);

SELECT *
FROM emp
WHERE hiredate = (SELECT max(hiredate) FROM emp);

Find seniors of BLAKE:

ENAME	HIREDATE

SELECT ename, hiredate
FROM emp
WHERE hiredate < (find BLAKE hiredate);

SELECT ename, hiredate
FROM emp
WHERE hiredate < (SELECT hiredate FROM emp
WHERE ename='BLAKE');

Find juniors of BLAKE:

ENAME	HIREDATE

SELECT ename, hiredate
FROM emp
WHERE hiredate > (find BLAKE hiredate);

SELECT ename, hiredate
FROM emp
WHERE hiredate > (SELECT hiredate FROM emp
WHERE ename='BLAKE');

Find the deptno which is spending max amount:

SELECT deptno
FROM emp
GROUP BY deptno
HAVING sum(Sal) = (find max amt in dept wise sum of sals);

find max amt in dept wise sum of sals:

SELECT max(sum(Sal))
FROM emp
GROUP BY deptno;

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 amt:

```
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));
```

Modify 7900 emp sal as 30th dept's max sal:

```
UPDATE emp
SET sal = (find 30th dept max sal)
WHERE empno=7900;
```

```
UPDATE emp
SET sal=(SELECT max(Sal) FROM emp WHERE deptno=30)
WHERE empno=7900;
```

Delete most senior record:

```
DELETE FROM emp
WHERE hiredate = (find most senior hiredate);
```

```
DELETE FROM emp
WHERE hiredate = (SELECT min(hiredate) FROM emp);
```

Multi row sub query:

- If sub query returns multiple rows then it is called "multi row sub query".
- For multi row sub query we cannot relational operators.
  We must use: IN, ANY, ALL.

Examples on multi row sub query:

Display the emp records whose job titles are same as SMITH and BLAKE:

ENAME	JOB	SAL

```
SELECT ename, job, sal
FROM emp
WHERE job IN(find SMITH and BLAKE job titles);
```

```
SELECT ename, job, sal
FROM emp
WHERE job =(SELECT job FROM emp
WHERE ename IN('SMITH', 'BLAKE'));
```

Display the emp records whose salary is more than BLAKE and ALLEN:

ENAME	SAL

```
SELECT ename, sal
FROM emp
WHERE sal > ALL(find BLAKE and ALLEN sals);
```

```
SELECT ename, sal
FROM emp
WHERE sal>ALL(SELECT sal FROM emp
WHERE ename IN('BLAKE', 'ALLEN'));
```

Display the emp records whose salary is more than BLAKE or ALLEN:

ENAME	SAL

```
SELECT ename, sal
FROM emp
WHERE sal>ANY(find BLAKE and ALLEN sals);
```

```
SELECT ename, sal
FROM emp
WHERE sal>ANY(SELECT sal FROM emp
WHERE ename IN('BLAKE', 'ALLEN'));
```

NOTE:

SAL>ALL(2850, 1600)	SAL>2850 AND SAL>1600
SAL>ANY(2850, 1600)	SAL>2850 OR SAL>1600

SAL	WHERE SAL>ALL(2850, 1600)	WHERE SAL>ANY(2850, 1600)
5000	T	T
4000	T	T
2450	F	T
1500	F	F

Inline view / Inline sub query:

- **If sub query is written in FROM clause then it is called "Inline view / Inline sub query".**
- **To control execution order of clauses we write sub query in FROM clause.**
- **It acts like table.**

Execution order:

FROM

WHERE
GROUP BY
HAVING
SELECT
DISTINCT
ORDER BY
OFFSET
FETCH

Syntax:

```
SELECT <column_list>
FROM (<sub query>)
WHERE <condition>;
```

Examples on Inline view:

Find 2nd max sal:

```
SELECT ename, sal,
Dense_rank() OVER(ORDER BY sal DESC) AS rank
FROM emp
WHERE rank=2;
```

Output:

ERROR: RANK invalid identifier

By default, We cannot use column alias in WHERE clause. Because, WHERE gets executed before SELECT.

If WHERE is executed after SELECT then we can use column alias in WHERE clause. That is why write SELECT query in FROM clause.

```
SELECT DISTINCT sal
FROM (SELECT ename, sal,
Dense_rank() OVER(ORDER BY sal DESC) AS rank
FROM emp)
WHERE rank=2;
```

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 value for n:

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;
```

Display top 3 salaried emps records:

```
SELECT *
FROM (SELECT ename, sal,
Dense_rank() OVER(ORDER BY sal DESC) AS rank
FROM emp)
WHERE rank<=3;
```

ROWNUM:

- **It is a pseudo column.**
- **It is used to apply row numbers to records.**

Examples on rownum:

Display all emp names and salaries.
Apply row numbers to them:

SNO	ENAME	SAL

```
SELECT rownum as sno, ename, sal
FROM emp;
```

Display all managers records. Apply row numbers To them:

SNO	ENAME	JOB	SAL

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

Display all columns and rows of emp table.
Apply row numbers to them:

```
SELECT rownum AS sno, e.* FROM emp e;
```

Examples on inline view:

Display 3rd row from emp:

```
SELECT *
FROM (SELECT rownum as rn, ename, sal
FROM emp)
WHERE rn=3;
```

Display 3rd row, 7th row and 10th row from emp:

```
SELECT *
FROM (SELECT rownum as rn, ename, sal
FROM emp)
WHERE rn IN(3, 7, 10);
```

Display 6th row to 10th row:

```
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".**
- **It acts like column.**

Syntax:

```
SELECT (<sub query>)
```

FROM <table_name>;

Examples on scalar sub query:

Display number of rows of emp table and dept table:
OUTPUT:

EMP	DEPT
14	4

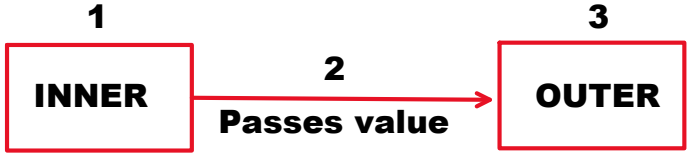
```
SELECT (select count(*) from emp) as emp,
(select count(*) from dept) as dept
FROM dual;
```

Non-correlated sub query:

- In this inner query passes value to outer query. But, outer query will not pass value to inner query.
- In this, inner query gets executed 1 time.

Execution process:

1. First inner query gets executed.
2. Inner query passes value to outer query.
3. Outer query gets executed.



Correlated Sub Query:

- If outer query passes value to inner query then it is called "Correlated sub query".

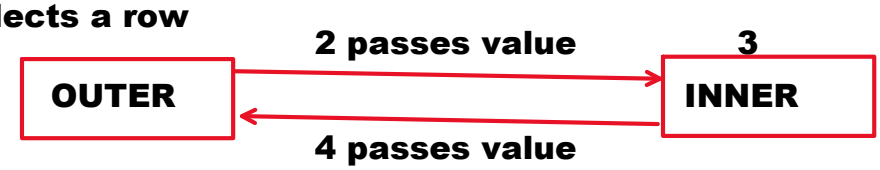
Execution process:

1. Outer query gets executed. 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 condn is

TRUE selects the row.

Above 5 steps will be executed repeatedly for every row selected by outer query.

1 selects a row



5 condn => T selects the row

Example on correlated sub query:

Display the emp records who are earning more than their dept's avrg salary:

Emp sal > Emp dept's avrg sal

```
SELECT ename, sal, deptno
FROM emp e
WHERE sal>(SELECT avg(Sal) FROM emp WHERE deptno=e.deptno);
```

EMP e

EMPNO	ENAME	DEPTNO	SAL
1001	A	10	10000
1002	B	20	20000
1003	C	30	30000
1004	D	10	30000
1005	E	20	10000
1006	F	30	50000

DEPTNO	AVG_SAL
10	20000
20	15000
30	40000

ENAME	DEPTNO	SAL
B	20	20000
D	10	30000
F	30	50000

Assignment:

Display the emps who are earning max sal in each dept:

Emp sal = emp dept's max sal

Assignment:
Display the seniors in each dept

Emp hiredate = emp dept's min hiredate

Rowid:

- **Rowid is a pseudo column.**
- **It is used to get address of row.**
- **It is used to deal with duplicate rows.**

Examples on row id:

Display dept table all rows along with row ids:

ROWID	DEPTNO	DNAME	LOC

```
SELECT rowid, d.* FROM dept d;
```

Example:

EMP16

EMPID	ENAME
1001	A
1001	A

Delete duplicate record:

```
DELETE FROM emp16
WHERE rowid='<type rowid here>';
```

Example on correlated sub query:

Delete duplicate rows:

EMP16

EMPID	ENAME	ROWID	Rowid!=min rowid in dup rows
1001	A	AAA	AAA != AAA FALSE
1001	A	AAB	AAB != AAA TRUE => delete
1002	B	AAC	AAC != AAC FALSE
1002	B	AAD	AAD != AAC TRUE => delete

```
DELETE FROM emp16 e
WHERE rowid!=(SELECT min(rowid) FROM emp16
WHERE empid=e.empid AND ename=e.ename);
```

Video link:
bit.ly/deletedups

EXISTS:

Syntax:

```
WHERE exists(<sub query>)
```

- **If sub query selects the rows then it returns TRUE.**
- **If sub query does not select the rows then it returns FALSE.**

Examples:

Display the dept names which are having emps:

```
SELECT dname
FROM dept d
WHERE exists(SELECT * FROM emp WHERE deptno=d.deptno);
```

Display the dept names which are not having emps:

```
SELECT dname
FROM dept d
WHERE not exists(SELECT * FROM emp WHERE deptno=d.deptno);
```

CONSTRAINTS

Wednesday, August 20, 2025 10:08 AM

CONSTRAINT:

GOAL:
It restricts the user from entering invalid data.
Maintaining valid data / accurate data / quality data.

CHECK(m1 BETWEEN 0 AND 100)

Max marks: 100
0 to 100

M1	NUMBER(3)
60	
50	
78	
123	error

CONSTRAINTS:

- **CONSTRAINT is a rule that applied on specific column.**
- **It is used to restrict the user from entering invalid data.**
- **Using it, we can maintain quality data / accurate data / valid data.**

- **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 it must be unique then use PK.**
- **A table can have only one PK.**

Example:

EMP

PK

EMPID	ENAME	JOB	SAL
1001	SAI	CLERK	20000
1002	KIRAN	CLERK	15000
error	NARESH	CLERK	10000
1001 error	A	MANAGER	30000

Example on PK:

T1

F1 INT PK

1

2

```
CREATE TABLE t1(f1 INT PRIMARY KEY);
Output:
Table created

INSERT INTO t1 VALUES(1);

INSERT INTO t1 VALUES(1);
--error:  unique constraint violated

INSERT INTO t1 VALUES(null);
--error:  cannot insert NULL into c##batch9am.T1.F1
```

NOT NULL:

- It does not accept nulls.
- It accepts duplicates.
- When value is mandatory and it can duplicated then use NOT NULL.

Example:

EMP

NOT NULL

EMPID	ENAME	SAL
1001	RAJU	12000
1002	RAJU	10000

1001	RAJU	12000
1002	RAJU	10000
1003		error 15000

UNIQUE:

- **It does not accept duplicates.**
- **It accepts nulls.**
- **When value is optional and it must be unique then use it.**

Example:

Customers

unique

CID	CNAME	MAIL_ID
123456	RAJU	raju@gmail.com
123457	KIRAN	raju@gmail.com error
123458	SRINU	
123459	RAVI	
123460	NARESH	naresh1234@yahoo.com

CONSTARINT	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.**

Syntax:

CHECK(<condition>)

Examples:

CHECK(m1 BETWEEN 0 AND 100)
M1 NUMBER(3)

Max marks: 100
0 TO 100

80
123 ERROR

CHECK(gender IN('M', 'F'))
GENDER CHAR

M
F
Z ERROR

Default:

- It is used to apply default value to column.
- When for most of the records value is same it is better to set that as default value.

Example:

STUDENT

DEFAULT 20000

SID	SNAME	FEE
1001	A	20000
1002	B	20000
1003	C	10000

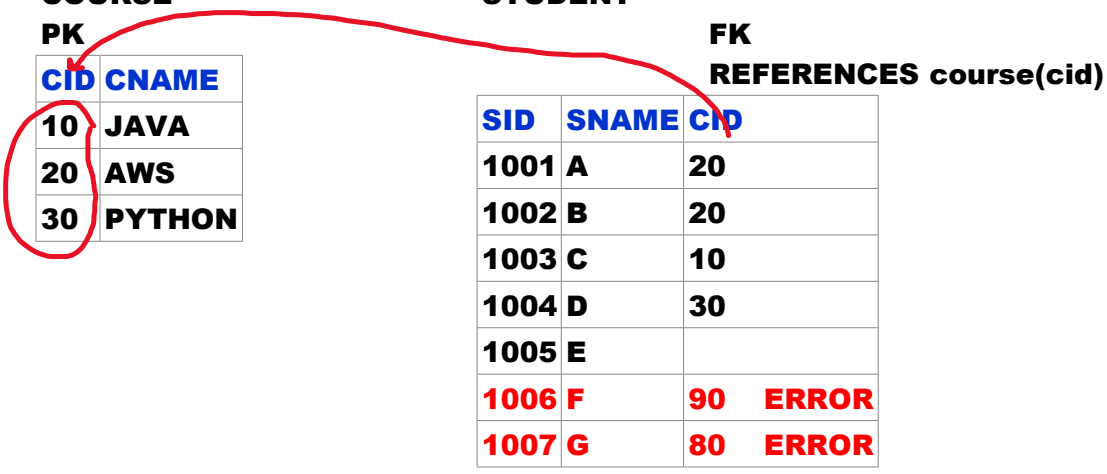
INSERT INTO student VALUES(1001, 'A');
Output:
ERROR: not enough values

INSERT INTO student(sid, sname) VALUES(1001, 'A');
INSERT INTO student(sid, sname) VALUES(1002, 'B');
INSERT INTO student VALUES(1003, 'C', 10000);

REFERENCES [FOREIGN KEY]:

- FK accepts PK values of another table.
- Using FK, we can establish relationship between 2 tables.
- It accepts duplicates.
- It accepts nulls.

Example:



Example:

STUDENT79

SID	SNAME	M1

SID	Don't accept dups and nulls	PK
SNAME	Value is mandatory. Accept duplicates.	NOT NULL
M1	Msust be b/w 0 to 100	CHECK

```
CREATE TABLE student79
(
  sid NUMBER(4) PRIMARY KEY,
  sname VARCHAR2(10) NOT NULL,
  m1 NUMBER(3) CHECK(m1 BETWEEN 0 AND 100)
);
```

Example:

USERINFO

USRID	UNAME	PWD

USRID	Don't accept dups and nulls	PK
Uname	Don't accept dups and nulls	UNIQUE NOT NULL
Pwd	Min 8 chars	CHECK(length(pwd)>=8)

```
CREATE TABLE userinfo
(
```

```
  usrid NUMBER(4) PRIMARY KEY,
  uname VARCHAR2(10) UNIQUE NOT NULL,
  pwd VARCHAR2(20) CHECK(length(pwd)>=8)
);
```

Assignment:

EMPLOYEE

EMPID	ENAME	GENDER	DOJ	SAL
PK	not null	check M / F	default sysdate	check >=5000

Example:

STUDENT125

PK NN default 'NARESH' default 'HYD' default 20000

SID	SNAME	CNAME	CCITY	FEE

```
CREATE TABLE student125
(
 sid NUMBER(4) PRIMARY KEY,
 sname VARCHAR2(10) NOT NULL,
 cname VARCHAR2(6) DEFAULT 'NARESH',
 ccity CHAR(3) DEFAULT 'HYD',
 fee NUMBER(7,2) DEFAULT 20000.00
);

INSERT INTO student125(sid, sname) VALUES(1001, 'A');
```

1001	A	NARESH	HYD	20000

```
INSERT INTO student125(sid, sname) VALUES(1002, 'B');
INSERT INTO student125(sid, sname) VALUES(1003, 'C');

COMMIT;

SELECT * FROM student125;
```

Example:

COURSE2

PK

CID	CNAME

STUDENT2

FK

REFERENCES course2(cid)

PK

CID	CNAME
10	JAVA
20	AWS
30	PYTHON

FK
REFERENCES course2(cid)

SID	SNAME	CID
1001	A	20
1002	B	30
1003	C	10
1004	D	10
1005	E	
1006	F	90 error

NOTE: PK and FK data types must be same

COURSE2
PK

CID	CNAME

```
CREATE TABLE course2
(
  cid NUMBER(2) PRIMARY KEY,
  cname VARCHAR2(10)
);
```

STUDENT2

FK
REFERENCES course2(cid)

SID	SNAME	CID

```
CREATE TABLE student2
(
  sid NUMBER(4),
  sname VARCHAR2(10),
  cid NUMBER(2) REFERENCES course2(cid)
);
```

Assignment:

DEPT2
PK

DEPTNO	DNAME
10	HR
20	SALES
30	ACCOUNTS

EMP2

FK
REFERENCES dept2(deptno)

EMPID	ENAME	DEPTNO
1001	A	30
1002	B	10

20	SALES
30	ACCOUNTS

1001	A	30
1002	B	10
1003	C	80 error

Examples:

ABHI BUS APP

```
CHECK(seat_num BETWEEN 1 and 40)
Seat_num
---------------
45     error
```

Max : 40 seats
1 to 40

UNIVERSITY

```
CHECK(medium IN('ENGLISH', 'TELUGU')
MEDIUM
--------------
ENGLISH
TELUGU
URDU        error
ARABIC      error
```

BANK

```
CHECK(t_type IN('W', 'D'))
T_TYPE
-------------
W
D
```

```
CHECK(mail_id  LIKE '%@%.@')
MAIL_ID
------------
  raju@gmail.com
  abcd                  error
```

Syntax to create the table:

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

Field definition:

<column> <data_type> CONSTRAINT <con_name> <con_type>

Naming Constraints

Thursday, August 21, 2025 9:51 AM

Naming Constraints:

- **To identify the constraint uniquely a name is required.**
  We can give names to constraints.
  As a developer we have to give constraint name.
  If we don't define constraint name implicitly ORACLE defines constraint name.
  Example: SYS_C123456 [oracle defined constraint name]

- **To disable or enable or drop the constraint name is required.**

Syntax to create the table:

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

Example:

STUDENT88

SID	SNAME	M1

PK CHECK => constraint types
C1 C2 => constraint names

```
CREATE TABLE student88
(
 sid NUMBER(4) CONSTRAINT c1 PRIMARY KEY,
 sname VARCHAR2(10),
```

```
 m1 NUMBER(3) CONSTRAINT c2 CHECK(m1 BETWEEN 0 AND 100)
);
```

User_Constraints:

- **It is a system table.**
- **It maintains all constraints information.**

To see constraints info:

```
DESC user_constraints

SELECT constraint_name, constraint_type, table_name
FROM user_constraints
WHERE table_name='STUDENT88';
```

NOTE:

- **We cannot give constraint name to DEFAULT.**

We can apply constraint at 2 levels.
They are:

- **Column level**
- **Table level**

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.**
  [PK, NOT NULL, UNIQUE, CHECK, DEFAULT, FK]

Example:

```
CREATE TABLE student88
(
 sid NUMBER(4) CONSTRAINT c1 PRIMARY KEY,
 sname VARCHAR2(10),
 m1 NUMBER(3) CONSTRAINT c2 CHECK(m1 BETWEEN 0 AND 100)
);
```

Table level constraint:

- **If constraint is defined after defining all columns then it is called "Table Level Constraint".**

- **Only 4 constraints can be applied at table level.**
  [PK, UNIQUE, CHECK, FK]

```
CREATE TABLE student89
(
 sid NUMBER(4),
```

```
 sname VARCHAR2(10),
 m1 NUMBER(3),
 CONSTRAINT c3 PRIMARY KEY(sid),
 CONSTRAINT c4 CHECK(m1 BETWEEN 0 AND 100)
);
```

Why Table level?

2 reasons:

- **To set combination of columns as constraint.**
- **To use another column name in constraint.**

setting combination of columns as constraint:

Example:

STUDENT91
PK(sid, subject)

SID	SNAME	SUBJECT	MARKS
1001	A	SUB1	70
1001	A	SUB2	60
1002	B	SUB1	60
1002	B	SUB2	90
1001		SUB1 error	
Null error			
		Null error	

STUDENT91
PK(sid, subject)

SID	SNAME	SUBJECT	MARKS

CREATE TABLE student91

```
(
 sid number(4),
 sname varchar2(10),
 subject char(4),
 marks number(3),
 CONSTRAINT c5 PRIMARY KEY(sid, subject)
);
```

If primary key is applied on combination of columns then it is called "Composite Primary key".

Using another column name in constraint:

PRODUCTS

PID	PNAME	MFRD_DATE	EXPIRY_DATE
1001	X	22-AUG-25	25-DEC-24

CHECK(expiry_date>mfrd_date)

```
CREATE TABLE products
(
 pid NUMBER(4),
 pname VARCHAR2(10),
 mfrd_Date DATE,
 expiry_Date DATE,
 CONSTRAINT c6 CHECK(expiry_date>mfrd_date)
);
```

Altering Constraints

Friday, August 22, 2025 10:39 AM

ALTER:

- **Using ALTER command we can:**
  - **Add the columns**
  - **Rename the columns**
  - **Drop the columns**
  - **Modify the field sizes**
  - **Modify the data types**

  - **Add the constraints**
  - **Rename the constraints**
  - **Drop the constraints**
  - **Disable the constraints**
  - **Enable the constraints**

Syntax:

```
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:

STUDENT19

SID	SNAME	M1

```
CREATE TABLE student19
(
  sid NUMBER(4),
  sname VARCHAR2(10),
  m1 NUMBER(3)
);
```

Add PK to sid:

```
ALTER TABLE student19
ADD CONSTARINT c10 PRIMARY KEY(sid);
(or)
ALTER TABLE student19
MODIFY sid CONSTRAINT c10 PRIMARY KEY;
```

Add NOT NULL to sname:

```
ALTER TABLE student19
MODIFY sname CONSTRAINT c11 NOT NULL;
```

Add check constraint to m1:

```
ALTER TABLE student19
ADD CONSTRAINT c12 CHECK(m1 BETWEEN 0 AND 100);
```

(OR)

```
ALTER TABLE student19
MODIFY m1 CONSTRAINT c12 CHECK(m1 BETWEEN 0 AND 100);
```

Rename c10 as z:

```
ALTER TABLE student19
RENAME CONSTRAINT c10 TO z;
```

Disable PK z:

```
ALTER TABLE student19
DISABLE CONSTRAINT z;
--temporarily PK will not work
```

Enable PK z:

```
ALTER TABLE stduent19
ENABLE CONSTRAINT z;
--again PK works
```

Drop PK z:

```
ALTER TABLE student19
DROP CONSTRAINT z;
--permanently PK z will be dropped
```

Cascade rules:

3 rules:

- **On delete no action [default]**
- **On delete cascade**
- **On delete set null**

On delete no action [default]:

parent table / master table	Child table / detailed table
T1	T2
PK	FK references T1(f1)
F1 INT	F1 INT
----------	----------
1	1
2	1
	2
	2

```
CREATE TABLE t1
(
 f1 INT,
 CONSTRAINT c15 PRIMARY KEY(f1)
);

CREATE TABLE t2
(
 f1 INT,
CONSTRAINT c16  FOREIGN KEY(f1) REFERENCES T1(f1)
);

--Insert records

DELETE FROM t1 WHERE f1=1;
Output:
ERROR:
```

By default,
We cannot delete parent record if it has child records.
[on delete no action]

If we want to delete parent record,
First delete all child records then only we can delete parent record.

On delete cascade:

If we delete parent record it also deletes child records automatically.

T3

PK
F1 INT

1
2

T4

FK
F1 INT

1
1
2
2

```
create table t3
(
f1 int constraint c20 primary key
);

Create table t4
(
 f1 int constraint c21 references t3(f1) on delete cascade
);
```

On delete set null:

If we delete parent record automatically it modifies child records as null.

T5

PK
F1 INT

1
2

T6

FK
F1
---
1 null
1 null
2
2

```
CREATE TABLE t5
(
 f1 INT CONSTRAINT c25 PRIMARY KEY
);

CREATE TABLE t6
(
```

```
 f1 INT CONSTRAINT c26 REFERENCES t5(f1) on delete set null
);
```

Copying Table

Saturday, August 23, 2025 10:31 AM

Copying Table:

Copying table means creating new table from existing table.

Syntax:

```
CREATE TABLE <table_name>
AS
<Select query>;
```

Example:

Create exact copy of emp table with the name emp1:

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

Example:

Create a new table with the name emp2
From existing table emp
With 4 columns empno, ename, job, sal
With managers rows:

EMP
8 cols
14 rows

EMP2
4 cols - empno, ename, job, sal
mgrs

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

Copying table structure

Saturday, August 23, 2025 10:44 AM

Copying table structure:

Syntax:

```
CREATE TABLE <table_name>
AS
SELECT <column_list>
FROM <table_name>
WHERE <false condition>;
```

False condition:

1=2
500=600
'A'='B'

Example:

Emp	EMp3
8 cols	8 cols
14 rows	no rows

Create new table with the name emp3
With emp table all cols
Without rows:

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

Copying Records

Monday, August 25, 2025 9:21 AM

Syntax:

```
INSERT INTO <table_name>[(<column list>)]
<SELECT query>;
```

Example:


```
CREATE TABLE emp1
AS
SELECT empno, ename, job, sal
FROM emp
WHERE 1=2;
```

Copy emp table all rows to emp1:

```
INSERT INTO emp1
SELECT empno, ename, job, sal FROM emp;
```

Output:

14 rows created.

INSERT ALL

Monday, August 25, 2025 9:32 AM

INSERT ALL:

- Introduced in ORACLE 9i.
- It is used to copy one table data to multiple tables.
- It avoids of writing multiple INSERT commands.
- It improves performance of copying records.

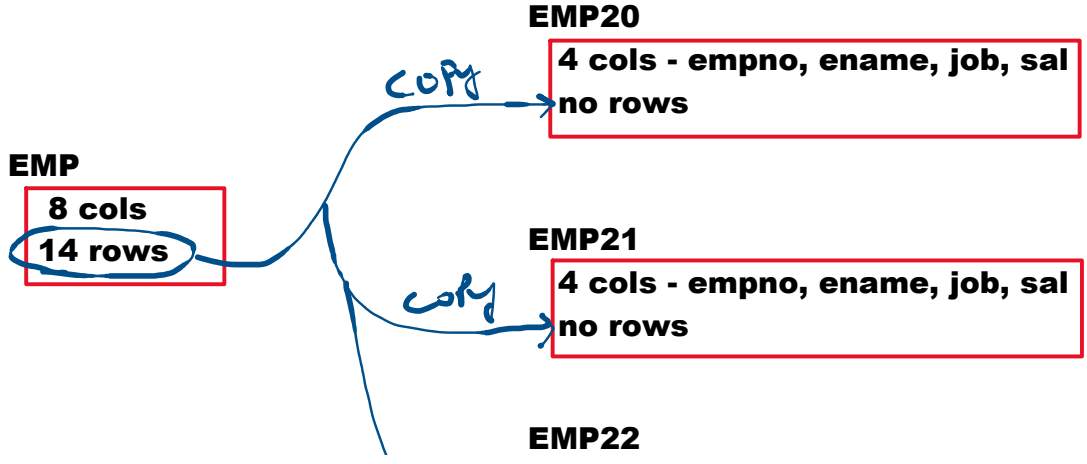
- It can be used to perform ETL operations.
  E => Extract T => Transfer L => Load

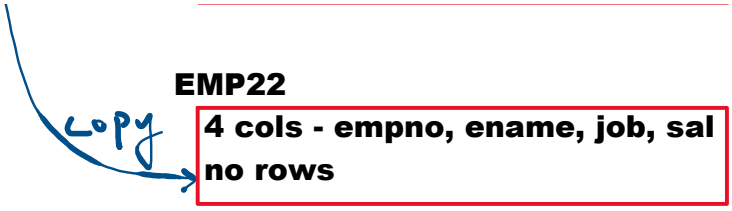
- It can be used in 2 ways. They are:
  - Unconditional INSERT ALL
  - Conditional INSERT ALL

Unconditional INSERT ALL:

```
INSERT ALL
INTO <table_name>(<column list>) VALUES(<value list>)
INTO <table_name>(<column list>) VALUES(<value list>)
.
.
<select query>;
```

Example:





Create emp20, emp21, emp22
With 4 cols
Without rows
From existing table emp:

```
CREATE TABLE emp20
AS
SELECT empno, ename, job, sal
FROM emp
WHERE 1=2;

CREATE TABLE emp21
AS
SELECT empno, ename, job, sal
FROM emp
WHERE 1=2;

CREATE TABLE emp22
AS
SELECT empno, ename, job, sal
FROM emp
WHERE 1=2;
```

Copy emp table all rows to emp20, emp21, emp22:

```
INSERT ALL
INTO emp20 VALUES(empno, ename, job, sal)
INTO emp21 VALUES(empno, ename, job, sal)
INTO emp22 VALUES(empno, ename, job, sal)
SELECT empno, ename, job, sal FROM emp;
```

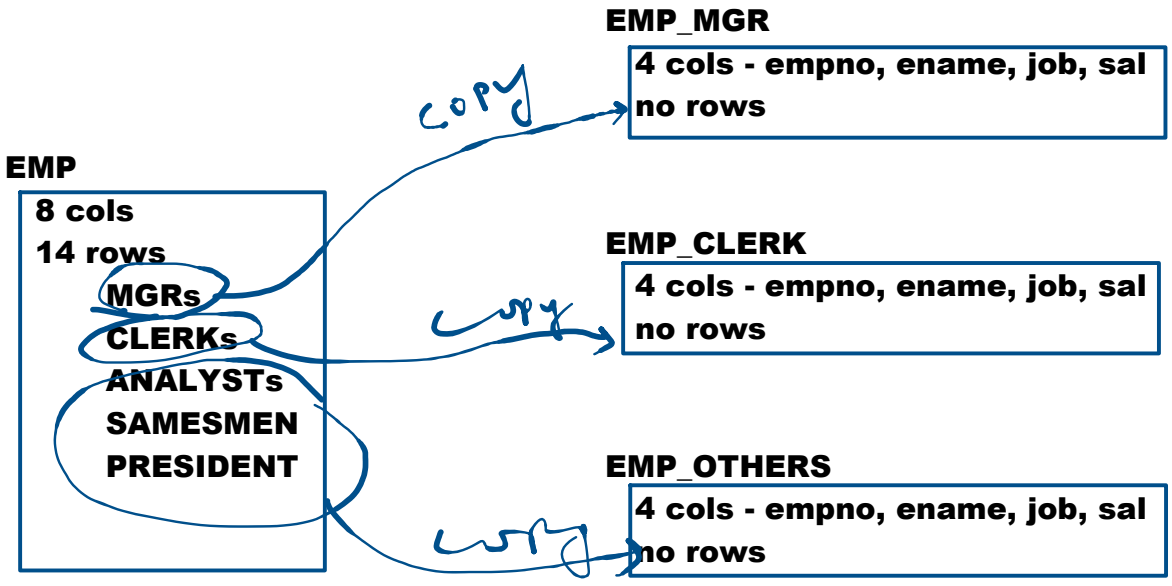
Output:
42 rows created.

```
COMMIT;
```

Conditional INSERT ALL:

```
INSERT ALL
WHEN <conditon1> THEN
   INTO <table_name>(<column list>) VALUES(<value list>)
WHEN <condition2> THEN
   INTO <table_name>(<column list>) VALUES(<value list>)
.
.
ELSE
   INTO <table_name>(<column list>) VALUES(<value list>)
<select query>;
```

Example:



Create emp_mgr, emp_clerk and emp_others tables
With 4 columns
Without rows
From existing table emp:

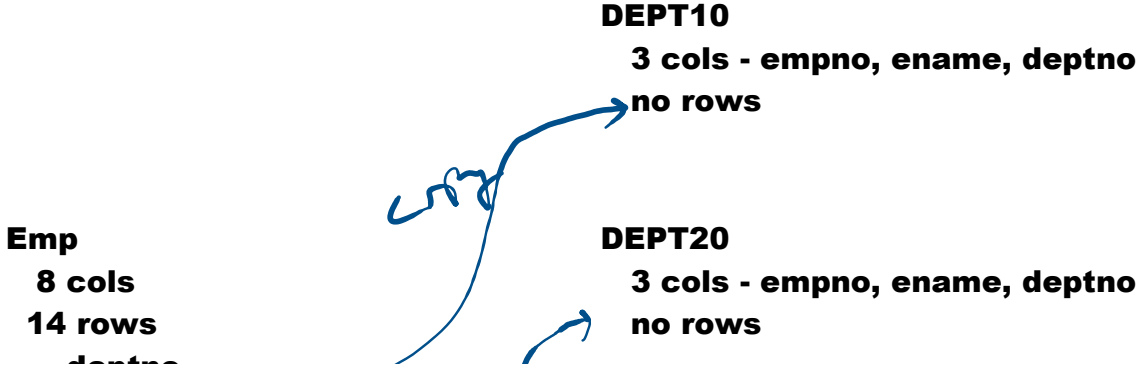
CREATE TABLE emp_mgr

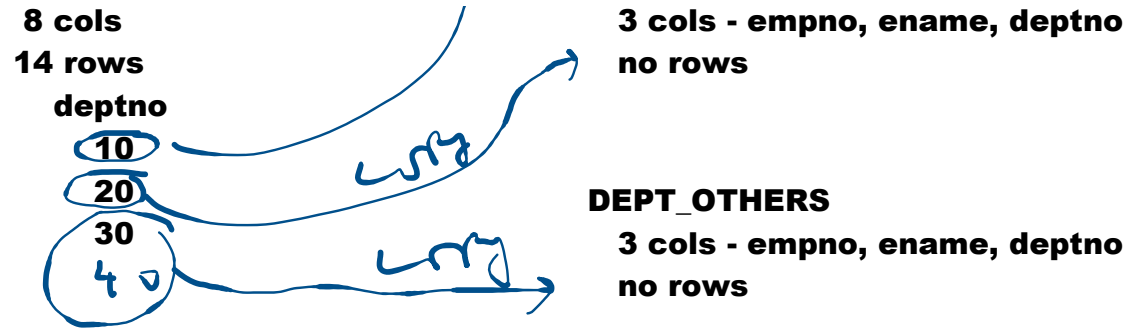
```
AS
SELECT empno, ename, job, sal
FROM emp
WHERE 1=2;

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 all managers rows to emp_mgr
clerks emp_clerk
others emp_others:

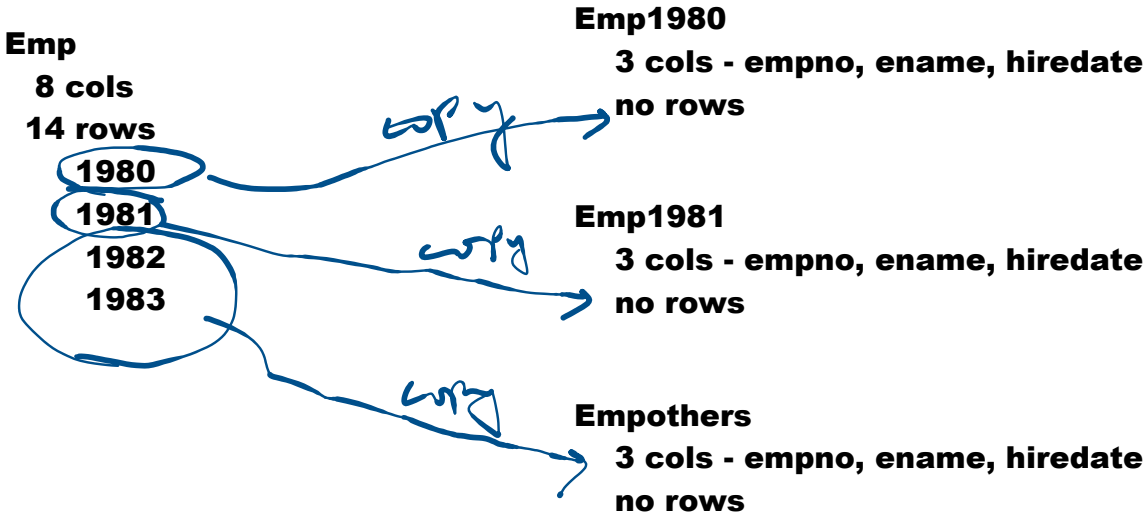
```
INSERT ALL
WHEN job='MANAGER' THEN
  INTO emp_mgr VALUES(empno, ename, job, sal)
WHEN job='CLERK' THEN
  INTO emp_clerk VALUES(empno, ename, job, sal)
ELSE
  INTO emp_others VALUES(empno, ename, job, sal)
SELECT * FROM emp;

COMMIT;
```

Assignment:





Assignment:




```
INSERT ALL
WHEN To_char(hiredate,'yyyy')=1980 THEN
  INTO emp1980 VALUES(empno, ename, hiredate)
.
.
```

MERGE

Monday, August 25, 2025 10:28 AM

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

CUSTOMER s

CID	CNAME	CCITY
1001	A AB	HYD BLR
1002	B	MUM
1003	C	CHN
1004	D	DLH

CUSTOMER_COPY t - Replica

CID	CNAME	CCITY
1001	A	HYD
1002	B	MUM

MERGE:

- Introduced in oracle 9i.
- MERGE command is used to apply one table changes to its REPLICA.
- MERGE = UPDATE + INSERT
- MERGE is a combination of UPDATE and INSERT commands.
- It can be also called as UPSERT command.

- If existing record => UPDATE
- If new record => INSERT

Syntax:

```
MERGE INTO <target_table_name> <t_table_alias>
USING <source_table_name> <s_table_alias>
ON(<merge_condition>)
WHEN matched THEN
UPDATE SET <t_column>=<s_column>, .....
WHEN not matched THEN
INSERT VALUES(<source table column names>);
```

Example on MERGE:

CUSTOMER s **CUSTOMER_COPY t - Replica**

CUSTOMER s

CID	CNAME	CCITY
1001	~~A~~ AB	~~HYD~~ BLR
1002	B	MUM
1003	C	CHN
1004	D	DLH

CUSTOMER_COPY t - Replica

CID	CNAME	CCITY
1001	A	HYD
1002	B	MUM

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

INSERT INTO customer VALUES(1001, 'A', 'HYD');
INSERT INTO customer VALUES(1002, 'B', 'MUM');
COMMIT;

CREATE TABLE customer_copy
AS
SELECT * FROM customer;
```

1003	C	CHN
1004	D	DLH

```
INSERT INTO customer VALUES(1003, 'C', 'CHN');
INSERT INTO customer VALUES(1004, 'D', 'DLH');
COMMIT;
```

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

```
UPDATE customer
SET cname='AB', ccity='BLR'
WHERE cid=1001;

COMMIT;
```

Apply customer table changes to its replica customer_copy:

```
MERGE INTO customer_copy t
USING customer 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:

4 rows merged

```
COMMIT;
```

DCL

Tuesday, August 26, 2025 9:21 AM

DCL:

- **DCL => Data Control Language.**
- **It deals with data accessibility.**

- **ORACLE SQL provides following DCL commands:**
  - **GRANT**
  - **REVOKE**

GRANT:

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

Syntax:

```
GRANT <privileges list>
ON <db_object_name>
TO <user list>;
```

Examples:

C##BATCH9AM
TABLE emp

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

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

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

```
GRANT insert, update, delete
ON emp
```

TO c##userA;

Grant 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, c##userC:

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

Grant read-only permission to all users:

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

Grant read-only permission on emp table to c##userA and allow him to GRANT permission to others:

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

NOTE:
WITH GRANT OPTION:
It allows GRANTEE to give permission to others

REVOKE:

- **It is used to cancel the permissions on DB objects from other users.**

Syntax:

```
REVOKE <privileges_list>
ON <db_object_name>
FROM <user_list>;
```

```
ON <db_object_name>
FROM <user_list>;
```

Examples:

Cancel Read-only permission on emp from c##userA:

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

Cancel DML permissions on emp from c##userA:

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

Cancel all permissions on emp from c##userA:

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

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

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

Example:

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

Login as DBA:

Username: system

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

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

CREATE USER c##userB
IDENTIFIED 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 INT, f2 CHAR); 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

INSERT INTO c##userA.t1
VALUES(3,'C');
Output:
ERROR: insufficient privileges
```

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

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

COMMIT;
```

```
SELECT * FROM t1;
Output:
F1   F2
----------
1    A
2    B
3    C
```

```
ALTER TABLE c##userA.t1
ADD f3 DATE;
Output:
ERROR: insufficient privileges
```

```
GRANT all
```

ON t1 TO c##userB;	
	ALTER TABLE c##userA.t1 ADD f3 DATE; Output: Table altered.
DESC t1 Output: NAME ---------- F1 F2 F3	
REVOKE all ON t1 FROM c##userB;	
	SELECT * FROM c##userA.t1; Output: Table does not exist

User_tab_privs_made
User_tab_privs_recd

User_tab_privs_made:

- It is a system table.
- It maintains all privileges info which are made by GRANTOR.

To see privileges info made by GRANTOR:

DESC user_tab_privs_made

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

User_tab_privs_recd:

- **It is a system table.**
- **It maintains privileges info which are received by GRANTEE.**

To see privileges info recieved by GRANTEE:

```
DESC user_tab_privs_recd

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

NOTE:
All => 12 permissions

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

SET OPERATORS

Thursday, August 28, 2025 9:24 AM

In Maths, SETS

A = {1,2,3,4,5}
B = {4,5,6,7,8}

A U B = {1,2,3,4,5,6,7,8}
A UA B = {1,2,3,4,5,4,5,6,7,8}
A I B = {4,5}

A M B = {1,2,3} => specific elements of A
Gives all elements from A except common elements

B M A = {6,7,8} => specific elements of B
Gives all elements from B except common elements

SET OPERATORS:

```
<SELECT query>
    <SET OPERATOR>
<SELECT query>;
```

- **SET OPERATOR is used to combine result of 2 select queries.**
- **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 with duplicates**
INTERSECT	**It gives common records from result of 2 select queries**
MINUS	**It gives specific records from 1st select query result.**

Example:

UNIVERSITY => SPORTS

CRICKET

SID	SNAME
1001	A
1002	B
1003	C

FOOTBALL

SID	SNAME
5001	D
1002	B
5002	E

Display the students who are participating in CRICKET and FOOTBALL:

```
SELECT * FROM cricket
UNION
SELECT * FROM football;
```

1001	A
1002	B
1003	C

5001	D
1002	B
5002	E

1001	A
1002	B
1003	C
5001	D
5002	E

Display the students who are participating in

CRICKET and FOOTBALL including duplicates:

```
SELECT * FROM cricket
UNION ALL
SELECT * FROM football;
```

1001	A
1002	B
1003	C

5001	D
1002	B
5002	E

1001	A
1002	B
1003	C
5001	D
1002	B
5002	E

Display the students who are participating in both CRICKET and FOOTBALL:

```
SELECT * FROM cricket
INTERSECT
SELECT * FROM football;
```

1001	A
1002	B
1003	C

5001	D
1002	B
5002	E

1002	B

Display the students who are participating in CRICKET only and those should not be participated in FOOTBALL:

```
SELECT * FROM cricket
MINUS
SELECT * FROM football;
```

1001	A
1002	B
1003	C

5001	D
1002	B
5002	E

1001	A
1003	C

Display the students who are participating in FOOTBALL only and those should not be participated in CRICKET:

```
SELECT * FROM football
MINUS
SELECT * FROM cricket;
```

5001	D
1002	B
5002	E

1001	A
1002	B
1003	C

5001	D
5002	E

Example:

EMP_IND

EMPID	ENAME
1001	A
1002	B
1003	C

EMP_US

EMPID	ENAME
5001	D
5002	E
5003	F

Display the emp records who are working for IND and US:

```
SELECT * FROM emp_ind
UNION
SELECT * FROM emp_us;
```

1001	A
1002	B
1003	C

5001	D
5002	E
5003	F

1001	A
1002	B
1003	C
5001	D
5002	E
5003	F

Example:

EMP_IND

EMPID	ENAME	DEPTID
1001	A	30
1002	B	10
1003	C	30

EMP_US

EMPID	ENAME	DEPTID
5001	D	20
5002	E	10
5003	F	30

DEPT

DEPTID	DNAME
10	HR
20	SALES
30	ACCOUNTS

Display emp details along with dept details who are working for IND and US:

ENAME	DNAME
A	ACCOUNTS
B	HR
C	ACCOUNTS
D	SALES
E	HR
F	ACCOUNTS

```
SELECT e.ename, d.dname
FROM emp_ind e, dept d
WHERE e.deptid=d.deptid
UNION
SELECT e.ename, d.dname
FROM emp_us e, dept d
WHERE e.deptid=d.deptid;
```

Example:

DEPTNO 20

ANALYST
CLERK
MANAGER

DEPTNO 30

CLERK
MANAGER
SALESMAN

Display the job titles offered by deptno 20 and 30:

```
SELECT job FROM emp WHERE deptno=20
UNION
SELECT job FROM emp WHERE deptno=30;
```

Display common job titles offered by deptno 20 and 30:

```
SELECT job FROM emp WHERE deptno=20
INTERSECT
SELECT job FROM emp WHERE deptno=30;
```

Display specific job titles of deptno 20 and those should not be offered by deptno 30:

```
SELECT job FROM emp WHERE deptno=20
MINUS
SELECT job FROM emp WHERE deptno=30;
```

Display specific job titles of deptno 30 and those should not be offered by deptno 20:

```
SELECT job FROM emp WHERE deptno=30
MINUS
SELECT job FROM emp WHERE deptno=20;
```

Differences b/w UNION and UNION ALL:

UNION	UNION ALL
• It does not give duplicates • Slower	• It gives duplicates • Faster

Differences b/w UNION and JOIN:

UNION	JOIN
• It combines rows. • It is used for horizontal merging.	• It combines columns. • It is used for vertical merging.

• It is applied on similar structures.	• It is applied on dissimilar structures.

Rules of SET OPERATORS:

- Number of columns in both select queries must be same.

Example:

```
SELECT sid, sname FROM cricket
UNION
SELECT sid FROM football;
Output:
ERROR
```

- Corresponding columns data types in both select queries must be same.

Example:

```
SELECT sid, sname FROM cricket
UNION
SELECT sname, sid FROM football;
Output:
ERROR
```

Module-1

Thursday, August 28, 2025 10:50 AM

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

Functions:

String functions
Conversion
Aggregate
Date
Number
Analytic
Other

Clauses:

SELECT
FROM
WHERE
GROUP BY
HAVING
DISTINCT
ORDER BY
OFFSET
FETCH

JOINS:

Inner Join => matched
- Equi
- Non-equi

Outer Join => matched + unmatched
- Left m + um from L
- Right m + um from R
- Full m + um from L and R

Self-Join
Cross Join

Sub queries

Non-correlated

- Single rows sq
- Multi row sq
- Inline view
- Scalar

Correlated sq

Constraints

PK
NOT NULL
UNIQUE
CHECK
DEFAULT
FK

SET OPERATORS

UNION
UNION ALL
INTERSECT
MINUS