Notes Link:
https://bit.ly/oracledbnotes

Akhil (Admin)
Mobile: 91541 56192 (Only Whatsapp)

ORACLE installation video link:
https://bit.ly/orainstall

ORACLE (SQL & PL/SQL) @ 6:00 PM (IST) By Mr. Shiva Chaitanya
Day-1 https://youtu.be/EKvNLAZnZAY
Day-2 https://youtu.be/23daYbbaFxI
Day-3 https://youtu.be/Z8i2or54gsw
Day-4 https://youtu.be/mSiIUPe0Iik
Day-5 https://youtu.be/28vk_rFPGEk
Day-6 https://youtu.be/9ddyRIfgi-g
Day-7 https://youtu.be/1ZlTUpMItoM
Day-8 https://youtu.be/0n3x0I6mqRk
Day-9 https://youtu.be/71rsNE7G39I

ORACLE syllabus

Monday, July 31, 2023 7:27 PM

SQL: Queries

SQL commands	**DDL, DRL, TCL, DML, DCL**
Built-In Functions	**String, Conversion, date, Analytical, Number**
CLAUSES	**Group By, Having, Order By**
JOINS	**types of joins**
SUB QUERIES	**types of sub queries**
SET OPERATORS	
VIEWS	
INDEXES	
SYNONYMS	
SEQUENCES	
MATERIALIZED VIEWS	

PL/SQL: Programs

PL/SQL Basics	**data types** **declare** **assign** **print** **read**
Control Structures	
Cursors	
Collections	
Exception Handling	
Stored Procedures	
Stored Functions	
Packages	

6 to 7.30 pm

21/2 to 3 months

Triggers	
Dynamic SQL	
Working with LOBs	

ORACLE

Monday, July 31, 2023 6:17 PM

Database
DBMS
RDBMS
Metadata

Database:

Bank DB	College DB
Branches	Courses
Products	Student
Customers	Fee
Transactions	Marks
Employee	Staff
	Libary

- **Database is a collection of interrelated data in an organized form.**
- **interrelated => Bank Db contains bank related data. Not college related data.**
- **organized => arranging in meaningful form**

DBMS:

- **DBMS => DataBase Management System / Software**
- **DBMS is a software that is used to create & maintain the database**

Evolution of DBMS:

Goal: storing business data in computer permanently

Before 1960s	Manually in BOOKS
In 1960s	FMS => File Management Software
In 1970s	HDBMS => Hierarchical DBMS NDBMS => Network DBMS
In 1976	RDBMS concept => E.F.Codd
In 1977	Larry Ellison => established ORACLE company with the name "Software Development Laboratories"
In 1979	RDBMS => ORACLE company name renamed as "Relational Software Inc."
In 1983	company name renamed as "ORACLE carp."

RDBMS:

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

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

Example:

3 columns
2 rows

Customer => Table / Relation / Entity

cid	cname	ccity
1001	AA	HYD
1002	BB	MUMBAI

Column / Attribute / Field / Property

Row / Tuple / Entity Instance / Record

Table	is a collection of rows and columns
Column / Field	is vertical representation of data holds individual values
Row / Record	is horizontal representation of data

	is a collection of field values

Metadata:

- **Data about the data is called "metadata".**
- **Metadata can be also called as "Data Definition".**

Examples:

field names	**empid, ename, salary**
table name	**EMPLOYEE**
data type	**NUMBER(4), VARCHAR2(10), DATE**
field size	**NUMBER(4) => 4 is field size** **VARCHAR2(10) => 10 is field size**

EMPLOYEE

empid **NUMBER(4)**	**ename** **VARCHAR2(10)**	**salary**
1234	**Ravi**	**5000**
SAI ERROR		
9999		
10000 => 5digits **ERROR**		

Database	**colln of interrelated data in an organized form**
DBMS	**is a software. to create & maintain the database**
RDBMS	**is a software.to create & maintain the database in the form of tables**
Metadata	**data about the data**

ORACLE

- **ORACLE is a Relational DataBase Management software [RDBMS].**
- **ORACLE is used to create & maintain the database in the form of tables.**
- **ORACLE Database s/w allows us to store, manipulate & retrieve the data of database.**

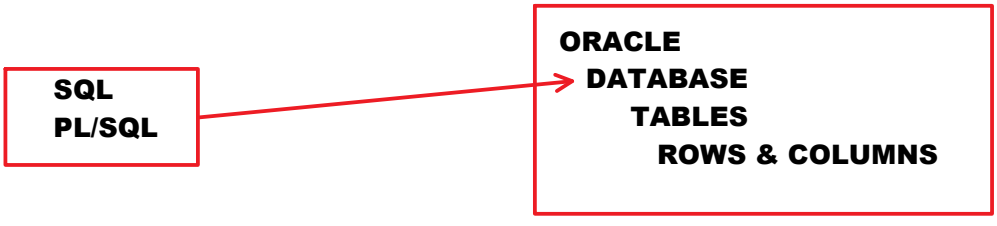
manipulate => 3 actions => INSERT / UPDATE / DELETE

new emp joined => INSERT
emp promoted => UPDATE [modify]
emp resigned => DELETE

retrieve => Opening existing data

Check balance
Searching for products
previous 3 months transactions

- **ORACLE software 2nd version introduced in 1979. They didn't release 1st version.**
- **Latest version for windows is: ORACLE 21C**



To communicate with ORACLE DATABASE we use 2 languages.

They are:

- SQL
- PL/SQL

SQL:

- SQL stands for Structured Query Language.
- SQL is a Query Language.
- It is used to write the queries.
- To communicate with ORACLE DATABASE, we write QUERIES in SQL.
- QUERY is a request that is sent to DB SERVER.
  Example:
  SELECT ename,sal FROM emp;
- SQL is Non-Procedural Language.
  Non-Procedural => no programs [no set of statements]

  In SQL we will not write any programs or a set of statements. Just, we write Queries.
- SQL is Unified Language.
  It is common language for many RDBMSs.

Java, C#, Python, C

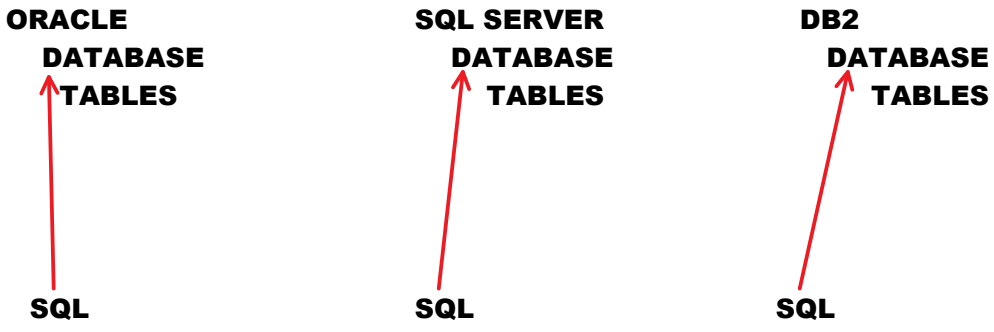
Programming Languages => develop the software
programs
instructions

Sub programs

In C:
Function:
a set of statements

In Java:
Method:
is a set of statements

In PL/SQL:
Procedure:
is a set of statements



PL/SQL:

- **PL / SQL => Procedural Language / Structured Query Language**
- **PL/SQL = SQL + Programming**
- **PL/SQL is a programming language. we develop the database programs to communicate with database.**
- **All SQL queries can be written as statements in PL/SQL program.**

if job is clerk => increase 1000 rupees salary to an emp
=> empno 7369
other wise increase 1500 salary

```
IF job='CLERK' THEN
   UPDATE emp SET sal=sal+1000 WHERE empno=7369;
ELSE
   UPDATE emp SET sal=sal+1500 WHERE empno=7369;
END IF;
```

SQL
PL/SQL

ORACLE
DATABASE
TABLES
Rows & Columns

SQL => query language => queries
non-procedural

PL/SQL => programming lang => programs
procedural

Bank Database

Customer

cid	cname	ccity	mobile	aadhar	pan

Transactions

Tid	T_date_time	Ttype	Acno	Amount

.
.

SQL Commands

Wednesday, August 2, 2023 6:15 PM

SQL provides 5 sub languages. They are:

DDL **[Data Definition Language]** • **Data Definition => metadata** • **it deals with metadata**	**CREATE** **ALTER** **DROP** **FLASHBACK [Oracle 10g]** **PURGE [Oracle 10g]** **TRUNCATE** **RENAME**
DRL / DQL **[Data Retrieval Language /** **Data Query Language]** • **Retrieve => opening existing data** • **It deals with data retrievals**	**SELECT**
TCL **[Transaction Control Language]** • **It deals with transactions**	**COMMIT** **ROLLBACK** **SAVEPOINT**
DML **[Data Manipulation Language]** • **Manipulation => 3 actions** **[INSERT / UPDATE / DELETE]** • **It deals with data manipulations**	**INSERT** **UPDATE** **DELETE** **INSERT ALL [Oracle 9i]** **MERGE [Oracle 9i]**
DCL **[Data Control Language]** • **It deals with data accessibility**	**GRANT** **REVOKE**

DDL:

CREATE:

EMPLOYEE

EMPNO	ENAME	SALARY

ALTER: Change

- **add the columns**
- **rename the columns**
- **drop the columns**
- **modifying field size**
- **modify data type**

ORACLE DB Objects

- **TABLE**
- **VIEW**
- **INDEX**
- **SEQUENCE**
- **SYNONYM**
- **MATERIALIZED VIEW**
- **STORED PROCEDURE**
- **STORED FUNCTION**
- **PACKAGE**
- **TRIGGER**

EMPLOYEE

EMPNO	ENAME	SALARY

DROP TABLE employee;

DROP	**used to delete the table dropped table goes to recyclebin**
FLASHBACK	**used to restore the table**
PURGE	**used to delete from recyclebin**

TRUNCATE	**used to delete entire table data with good performance**

RANAME

EMPLOYEE

EMPNO	ENAME	SALARY
1001		
1002		
..		
1100		

TCL:
Transaction Control Language

Transaction => is a series of actions [SQL commands]

Examples:
Withdraw, Deposit, Fund transfer, placing order
Taking Admission

Withdraw

Enter PIN: 1234

Enter Amount: 10000
machine gives money

ACCOUNT

ACNO	PIN	BALANCE
123456	1234	50000

Enter Amount: 10000
machine gives money
UPDATE

Transaction must be successfully finished (or) cancelled

DDL commands

Thursday, August 3, 2023 6:16 PM

CREATE:

- used to create the the database objects like tables, vies, indexes ....etc.

Syntax to create the table:

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

[ ]	Optional
< >	Any

Data Types in ORACLE SQL:

Data Type tells,

- How much memory has to be allocated
- Which type of data should be accepted
- valid range [domain]

ORACLE SQL provides following data types:

Character Related 'raju'	Char(n) Varchar2(n) LONG CLOB nChar(n) nVarchar2(n) nCLOB
Number Related 1234 6000.00	NUMBER(p) NUMBER(p,s)
Date & Time Related 25-DEC-22 25-DEC-22 10:30:0.0 AM	DATE TIMESTAMP
Binary Related images, audios, videos	BFILE BLOB

Character Related Data Types:

Char(n):

- n => field size [max no of chars]
- It is used to hold a set of characters [string]
- It is Fixed Length data type
- max size: 2000 bytes [2000 chars]
- default size: 1

Varchar2(n):

- n => field size [max no of chars]
- It is used to hold a set of characters [string]
- It is variable length data type
- max size: 4000 bytes [4000 chars]
- default size: no default size

State_Code CHAR(2)	ename VARCHAR2(10)
TS	arun
AP	vijay
WB	sai
MH	naresh

Vehicle_Number CHAR(10)	mail_id VARCHAR2(30)
TS09AA1234	abcd@gmail.com
	abcd1234@gmail.com

PAN_NUMBER CHAR(10)	pname VARCHAR2(20)
ABC3L1234Z	mouse
	micro processor

Note:

- VARCHAR2(n) data type can hold max of 4000 chars.
- To hold more than 4000 chars, we can use LONG and CLOB

LONG:

- used to hold large amounts of characters
- max size: 2GB
- It has some restrictions:
  - In one table, we can create 1 column as LONG type.
  - We cannot use built-in functions on LONG type.

CLOB:

- **CLOB => Character Large Object**
- **used to hold large amounts of characters**
- **max size: 4GB**
- **In one table, we can create any no of columns as CLOB type.**
- **We can use built-in functions on CLOB type**

Examples:

Experience_summary CLOB

Product_features CLOB

Feedback CLOB

Complaints CLOB

Comments CLOB

Character Related	**Char(n)** **Varchar2(n)** **LONG** **CLOB**	**•ASCII code char data types** **•can hold english lang chars only** **•Single Byte char data types**
National Character Related	**nChar(n)** **nVarchar2(n)** **nCLOB**	**•UNI code char data types** **•can hold english + other lang chars** **•Multi Byte char data types**

nChar(n)	**Fixed length data type** **max size: 2000 bytes [1000 chars]**
nVarchar2(n)	**Variable length data type** **max size: 4000 bytes [2000 chars]**
nCLOB	**max size: 4GB**

Number Related Data Types:

NUMBER(p):
- **It is used to hold integers.**
- **p => precision [max no of digits]**
- **p => valid range => 1 TO 38**

56	**integer:** **number without decimal places**
56.789	**floating point:** **number with decimal places**

Examples:

empid NUMBER(4) -9999 TO 9999

1234

1235
1236
123
12
6789
9999
10000 => ERROR
RAMU => ERROR

customer_id NUMBER(6) => -999999 TO 999999

912345

maths_marks NUMBER(3) => -999 TO 999

78
100
123
999
1000 => ERROR

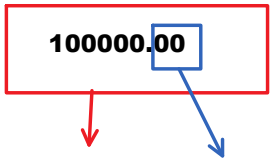
Aadhar_Number NUMBER(12)

Credit_Card_Number NUMBER(16)

Mobile_number NUMBER(10)

NUMBER(p,s):
- It is used to hold floating point values.
- p => precision [max no of digits]
- s => scale [max no of decimal places]

Example:



p=8 s=2

sal NUMBER(8,2) -999999.99 TO 999999.99

1000000 => ERROR

100.00

avrg NUMBER(5,2) => -999.99 TO 999.99

78.67
123.56
999.99
1000 => ERROR
123.67892345 => accepts value => 123.68
123.67489762 => accepts value => 123.67

height NUMBER(2,1) => -9.9 TO 9.9

5.0
5.3
5.8
5.4
10 => ERROR

Date & Time Related Data Types:

DATE:

- it is used to hold date values.
- Default oracle date format is: DD-MON-RR
- Date also contains time value.
- Date value contains day, month, year, hours, minutes and seconds.
- Default time is: 12:00:00 AM [midnight time]
- Date data type cannot hold fractional seconds.
- Fixed length data type
- size: 7 Bytes

Examples:

```
Date_Of_Birth DATE
Date_Of_Joining DATE
Date_Of_Retirement DATE
ordered_date DATE
```

Timestamp:

- introduced in Oracle 9i version.
- It is extension of DATE type.

- **it can hold day, month, year, hours, minutes, seconds and fractional seconds.**
- **memory: 11 Bytes**
- **Fixed length data type.**

Note:
DATE type is used to hold date values.

Example:
Transaction_date DATE

TIMESTAMP type is used to hold date and time values

Example:
Ordered_date_time TIMESTAMP

fixed length **variable length**

T1

F1 CHAR(10)	F2 VARCHAR2(10)
raju6spaces	raju
naresh4spaces	naresh
sai7spaces	sai

10
10
10

4
6
3

In c: ASCII coding system
char ch; //1 Byte

In Java: UNI coding system
char ch; //2 Bytes

ASCII coding system:

- **256 chars => 0 to 255**
- **Letters [A to Z, a to z]**
- **Digits [0 to 9]**
- **Special chars [+ $ %]**

255	1111 1111 => 1 byte

supports English lang only

A	65
B	66
..	
Z	90

a	97
b	98
..	
z	122

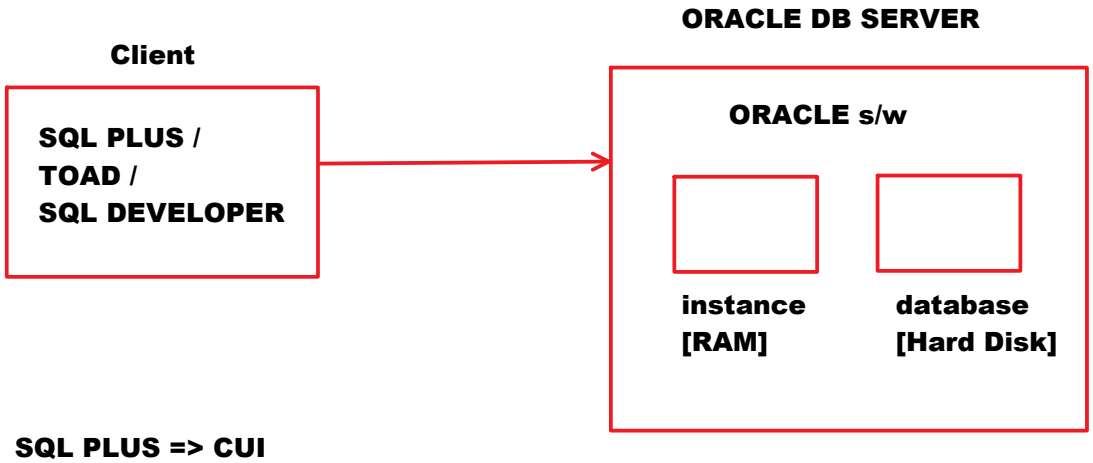
0	48
1	49
.	
9	57

UNI coding system:

- **65536 chars => 0 to 65535**

65535	**1111 1111 1111 1111 => 2 bytes**

- **supports english + other lang chars**



SQL PLUS => CUI
TOAD / SQL DEVELOPER => GUI

Opening SQL PLUS:

- **Press Windows + R . Displays RUN dialog Box.**
- **Type "sqlplus" and click on OK.**

Login DBA [DataBase Administrators]:

username: system
password: nareshit [at the time of ORACLE installation you have given one password in step4]

Creating User:

Syntax:

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

GRANT create session, create table, unlimited tablespace
TO <username>;
```

Example:

create one user with the username c##userA and password is usera:

Login as DBA:

common user	c##raju
local user	raju

common user	c##raju
local user	raju

Login as DBA:

username: system
password: nareshit [at the time of ORACLE installation you have given password]

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

GRANT create session, create table, unlimited tablespace
TO c##userA;
```

create session / connect	is a permission for log in
create table / resource	is a permission for creating table
unlimited tablespace	is a permission to use the memory [to insert the records]

Note:
SQL> show user
--display current user name

conn[ect]:

- **it is used to log in from SQL command prompt.**
- **Log in means, connecting to server.**
- **conn[ect] command is used to connect to the server.**

Syntax:

CONN[ECT] username/password

Example:

SQL> conn c##batch6pm/nareshit
Output:
Connected

Example:

username	c##batch6pm
password	nareshit

Login as DBA:

```
CREATE USER c##batch6pm
IDENTIFIED BY nareshit;
```

Output:
User created.

GRANT create session, create table, unlimited tablespace
TO c##batch6pm;

Examples on Creating Tables

Tuesday, August 8, 2023 6:13 PM

Syntax of Creating table:

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

For Windows OS:
ORACLE 21C => 1000 columns

For LINUX OS:
ORACLE 23C => 4000+ columns

[ ]	Optional
< >	Any

Syntax of INSERT Command:

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

Example-1:

STUDENT

sid	sname	avrg

sid	NUMBER(4)
sname	VARCHAR2(10)
avrg	NUMBER(5,2)

Creating Table:

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

Output:
Table created.

Inserting Records:
STUDENT

sid	sname	avrg
1001	AA	67.8
1002	BB	56.7

```
INSERT INTO student
VALUES(1001,'AA',67.8);      --inserts in RAM

INSERT INTO student
VALUES(1002,'BB',56.7);     --inserts in RAM

COMMIT;                              --data in RAM will be moved to HARD DISK
                                     --saves the data
```

To see table data:

```
SELECT * FROM student;
```

*	All Columns

```
*  = sid, sname, avrg
```

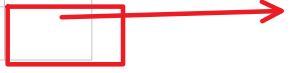
Implicitly ORACLE rewrites above query as following:

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

Inserting Limited Column Values:

STUDENT

sid	sname	avrg
1003	CC	



NULL => Empty / Blank

INSERT INTO student VALUES(1003,'CC');
Output:
ERROR: Not Enough Values

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

Inserting Limited Column Values By changing order of columns:

SID	SNAME	AVRG
1004	DD	

INSERT INTO student(sname,sid)
VALUES('DD',1004);

SID	SNAME	AVRG
1005		78.9

INSERT INTO student(sid,avrg) VALUES(1005,78.9);

Inserting Records using Parameters:

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

Syntax:
&<parameter_name>

<parameter_name>	any text

With this parameter name a value will be asked at run time

Parameter name need not be column name. Any text can be given.

Examples:

&sid

enter value for sid:

&A
enter value for A:

5001
5002
5003
.
.
5010

INSERT INTO student VALUES(&sid,'&sname',&avrg);
Output:
enter value for sid: 5001
enter value for sname: SS
enter value for avrg: 89.2
INSERT INTO student VALUES(&sid,'&sname',&avrg);
INSERT INTO student VALUES(5001,'SS',89.2);
1 row created.

/
Output:
enter value for sid: 5002
enter value for sname: TT
enter value for avrg: 66.4
1 row created

/
Output:
enter value for sid: 5003
enter value for sname: ZZ
enter value for avrg: 44.5
1 row created

Displaying Table Structure:

- **DESC[RIBE]:**
  - **DESC[RIBE] command is used to display table structure**

Syntax:

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

Example:

```
DESC student
```

Output:

```
NAME       TYPE
-----------------------------------------
sid            NUMBER(4)
sname       VARCHAR2(10)
avrg          NUMBER(5,2)
```

to see all tables list which are created by user:

USER_TABLES:

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

```
SELECT table_name FROM user_tables;
```

Output:

```
TABLE_NAME
------------------------
T1
STUDENT
```

RUN	**R**	**/**

R[UN]:

- **It is used to run recent command which is in memory**
- **It runs above query**

DOJ	Date_Of_Joining

Example-2:

EMPLOYEE

EMPNO	ENAME	GENDER	SAL	DOJ
1001	AA	M	8000	25-DEC-2022

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

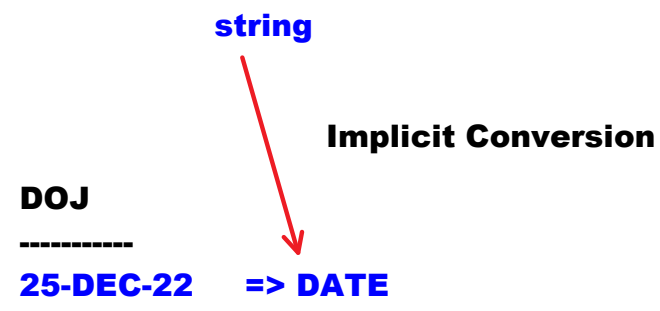
```
CREATE TABLE employee
(
empno NUMBER(4),
ename VARCHAR2(10),
gender CHAR(1),
sal NUMBER(8,2),
doj DATE
);
```

Output:
Table created.

1001	AA	M	8000	25-DEC-22

```
INSERT INTO employee VALUES(1001,'AA','M',8000,'25-DEC-2022');
```

Output:
1 row created



Note:

- **Don't depend on Implicit Conversion.**
- **Implicit Conversion degrades the performance.**

to_Date():

- **"to_Date()" is a built-in function.**
- **It converts string to date.**

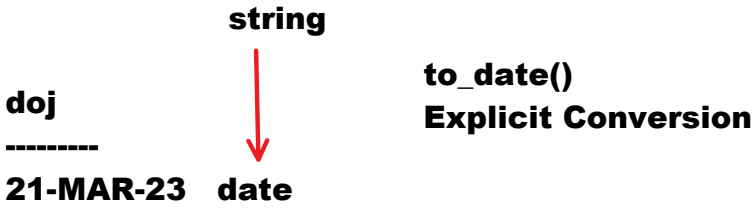
Example:

to_Date('25-DEC-2022') **string**	**25-DEC-22** **date**

Inserting employee record with following data. Use to_date() function to insert date value:

1002	**BB**	**F**	**7000**	**21-MAR-2023**

```
INSERT INTO employee
VALUES(1002,'BB','F',7000,to_Date('21-MAR-2023'));
```



Insert a record with today's date:

```
INSERT INTO employee
VALUES(1003,'CC','M',6000,sysdate);
```

sysdate	**•is a built-in function** **•it returns current system date** **to see current system date:**

```
SELECT sysdate FROM dual;
```

Example-3:

EMP

EMPNO	ENAME	LOGIN_DATE_TIME
1001	AA	27-JUL-23 10:30:0.0 AM

EMPNO	NUMBER(4)
ENAME	VARCHAR2(10)
LOGIN_DATE_TIME	TIMESTAMP

```
CREATE TABLE emp
(
empno NUMBER(4),
ename VARCHAR2(10),
login_date_time TIMESTAMP
);
```

Output:
Table created

1001	AA	27-JUL-23 10:30:0.0 AM	12:00:0.0 AM

```
INSERT INTO emp
VALUES(1001,'AA','27-JUL-2023 10:30:0.0 AM');
```

string

Implicit Conversion

```
login_date_time
---------------------------
27-JUL-23 10:30:0.0 AM      TIMESTAMP
```

to_Timestamp():

- **is a built-in function.**
- **used to convert string to timestamp**

Example:

to_Timestamp('25-DEC-2022 10:30:0.0 AM') **string**	**25-DEC-2022 10:30:0.0 AM** **timestamp**

```
INSERT INTO emp
VALUES(1001,'AA',to_timestamp('27-JUL-2023 10:30:0.0 AM'));
```

string

to_timestamp()
Explicit Conversion

```
login_date_time
-----------------------
27-JUL-23 10:30:0.0 AM        timestamp
```

```
INSERT INTO emp
VALUES(1234,'DD',systimestamp);
```

systimestamp	**is a built-in function** **it returns current system date and time** **displaying current system date and time:** **SELECT systimestamp FROM dual;**

Setting pagesize and linesize:

SQL> show all
--displays all parameters

pagesize	**14**
linesize	**80**

Setting pagesize:

SQL> SET PAGES 200
-- in one ORACLE page, it can display 200 lines

Setting linesize:

SQL> SET LINES 200
--in one line, it can display 200 chars

setting page size and line size:

SQL> SET PAGES 200 LINES 200

PAGES[IZE]	**used to set page size**
LINES[IZE]	**used to set line size**
CL[EAR] SCR[EEN]	**used to clear the screen**

DRL

Thursday, August 10, 2023 6:16 PM

DRL / DQL:

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

- **It mainly deals with data retrievals**

- **Retrieve => Opening existing data**

Example:
Checking Balance
Searching for products

ORACLE SQL provides only 1 DRL command.
That is:
- **SELECT**

SELECT:
- **used to retrieve [select] the data from table.**

Syntax:

```
SELECT [ALL/DISTINCT] <column_list / *>
FROM <table_name>
[WHERE <condition>]
[GROUP BY <grouping column_list>]
[HAVING <group_condition>]
[ORDER BY <column> ASC/DESC];
```

SQL	ENGLISH
QUERIES	SENTENCES
CLAUSES	WORDS

CLAUSE	• is a part of query • every CLAUSE has specific purpose • every CLAUSE performs specific action

- **used to retrieve [select] the data from table.**

- **Using SELECT command we can SELECT:**
  - **All Columns and All Rows**
  - **All Columns and Specific Rows**
  - **Specific Columns and All Rows**
  - **Specific Columns and Specific Rows**

SELECT *	**All Columns**
NO WHERE CLAUSE	**All Rows**
SELECT ename,sal	**Specific Columns**
WHERE CLAUSE	**Specific Rows**

SELECT clause	**used to specify column list**
FROM clause	**used to specify table name**

WHERE clause	used to specify filter condition

- **All Columns and All Rows:**

Display all emp records and all columns from emp table:

```
SELECT *
FROM emp;
```

*	All Columns

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

Note:
Implicitly * will be replaced with all column names

Above query will be rewritten by ORACLE as following:

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

All column names will be taken in table structure order

All Columns and specific rows:

Display the emp records whose salary is 3000:

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

- **Specific Columns and All Rows:**

Display the emp names and salaries of all emps:

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

- **Specific Columns and Specific Rows:**

Display emp names and salaries of the emps whose salary is 3000:

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

Execution Order:

FROM
WHERE
SELECT

EMP

EMPNO	ENAME	JOB	SAL
1001	A	CLERK	2000

1002	B	MANAGER	5000
1003	C	CLERK	3000
1004	D	SALESMAN	2500
1005	E	SALESMAN	3000

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

FROM emp:
It selects entire table

EMP

EMPNO	ENAME	JOB	SAL
1001	A	CLERK	2000
1002	B	MANAGER	5000
1003	C	CLERK	3000
1004	D	SALESMAN	2500
1005	E	SALESMAN	3000

Result of FROM clause

WHERE sal=3000:

- **WHERE clause filters the rows**
- **WHERE clause condition will be applied on every row**
- **If condition is TRUE, row will be selected.**
- **If condition is FALSE, row will not be selected.**

EMP

EMPNO	ENAME	JOB	SAL
1001	A	CLERK	2000
1002	B	MANAGER	5000
1003	C	CLERK	3000
1004	D	SALESMAN	2500
1005	E	SALESMAN	3000

WHERE sal=3000
2000 = 3000 FALSE
5000 = 3000 FALSE
3000 = 3000 TRUE
2500 = 3000 FALSE
3000 = 3000 TRUE

EMP

EMPNO	ENAME	JOB	SAL
1003	C	CLERK	3000
1005	E	SALESMAN	3000

Result of WHERE clause

SELECT ename,sal:
It selects specified columns

EMP

EMPNO	ENAME	JOB	SAL
1003	C	CLERK	3000
1005	E	SALESMAN	3000

ENAME	SAL
C	3000
E	3000

RESULT OF SELECT

OPERATORS in ORACLE SQL:

OPERATOR:

- **OPERATOR is a symbol that is used to perform operations like arithmetic operations or logical operations.**

ORACLE SQL provides following Operators:

Arithmetic	**+ - * /**
Relational / Comparison	**> < >= <= = != / <> / ^=** **equals not equals**
Logical	**AND OR NOT**
Special	**IN** **NOT IN** **BETWEEN AND** **NOT BETWEEN AND** **LIKE** **NOT LIKE** **IS NULL** **IS NOT 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:

+	**Addition**
-	**Subtraction**
*****	**Multiplication**
/	**Division**

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

In SQL:
MOD(5,2) => 1 [remainder]
MOD => function

5/2 = 2.5

Examples on Arithmetic Operators:

Calculate Annual Salary of all employees:

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

Output:

```
ENAME          SAL       SAL*12
-----------    ----------  ----------------
SMITH           800         9600
ALLEN          1600        19200
```

Column Alias:

- **Column Alias is used to change column heading.**
- **Column alias is temporary.**
- **Its scope is limited to that query only. Column Alias cannot be used in another query.**
- **"AS" keyword can be used to give column alias**
- **Using "AS" keyword is optional**
- **To maintain the case or to give alias name in multiple words specify alias name in double quotes.**

Syntax:

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

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

Output:

```
ENAME          SAL              ANNUAL_SAL
---------------  -------------  ----------------
```

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

Output:

ERROR

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

Output:

```
ENAME          SAL              annual salary
---------------  -------------  ----------------
```

```
SELECT ename, sal, sal*12 AS "ANNUAL SALARY"
FROM emp;
```

Output:

```
ENAME          SAL              ANNUAL SALARY
---------------  -------------  ----------------
```

Calculate experience of all employees:

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

Calculate TA, HRA, TAX and GROSS SALARY of all emps.
10% on sal as TA
20% on sal as HRA
2% on sal as TAX
GROSS = SAL + TA + HRA - TAX

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

Relational Operators / Comparison Operators:

- **Relational Operators are used compare column value with 1 value.**

ORACLE SQL provides following Relational Operators:

>	greater than
<	less than
>=	greater than or equals to
<=	less than or equals to
=	equals to
!= / <> / ^=	not equals to

Syntax:

```
WHERE <column> <relational_operator> <value>
```

Examples:

```
WHERE sal=3000
WHERE sal>3000
WHERE sal!=3000
```

Examples on Relational Operators:

Display the emp records whose salary is more than 2500:

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

Display the emp records whose salary 1250 or less:

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

Note:
Calendar Order is ASCENDING ORDER [small to big]

1-JAN-2022
2-JAN-2022
3-JAN-2022
. Who joined after 2022
.
31-DEC-2022
1-JAN-2023
2-JAN-2023
. → >'31-DEC-2022'
.
31-DEC-2023

Display the emp records who joined after 1981:

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

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

1-JAN-1982 onwards

Display the emp records who joined before 1981:

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

.
.
30-DEC-1980 → hiredate<'1-JAN-1981'
31-DEC-1980
1-JAN-1981
.

.
31-DEC-1981

Display the emp record whose empno is 7900:

```
SELECT *
FROM emp
WHERE empno=7900;
```

Display all managers records:

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

MANAGER = manager FALSE

Output:
no rows selected

Note:

- SQL is not case sensitive language,
- String comparison is case sensitive.
  Lower case and Upper case will be treated as different.

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

Output:
displays all managers records

Display the emp records who are working in deptno 30:

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

Display all emps records except managers:

```
SELECT empno,ename,job,sal
FROM emp
WHERE job!='MANAGER';
(or)
SELECT empno,ename,job,sal
FROM emp
WHERE job<>'MANAGER';
(or)
SELECT empno,ename,job,sal
FROM emp
WHERE job^='MANAGER';
```

Logical Operators:

Logical Operators are used to perform Logical Operations like Logical AND, Logical OR, Logical NOT operations.

ORACLE SQL provides 3 Logical Operators. They are:

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

- **To separate multiple conditions we can use AND, OR operators.**

All Conditions should be satisfied	**AND**
At least 1 Condition should be satisfied	**OR**

Truth Table:

c1	**condition1**
c2	**condition2**

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

AND	**All T => T**
OR	**Min 1 T => T**

Examples on logical operators:

Display all managers and clerks records:

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

Display the emp records whose salary is 2450 or more, and 3000 or less [whose salary is between 2450 and 3000]

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

Display the emp records whose empnos are 7521, 7698, 7900:

```
SELECT *
FROM emp
WHERE empno=7521 AND empno=7698 AND empno=7900;
```

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

```
SELECT empnoe,name,deptno
FROM emp
WHERE deptno=10 OR deptno=30;
```

Display the emp records who joined in 1982:

after 1982	hiredate>'31-DEC-1982'
before 1982	hiredate<'1-JAN-1982'

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

Display the emp records whose names are BLAKE, JAMES and WARD:

```
SELECT *
FROM emp
WHERE ename='BLAKE' OR ename='JAMES' OR ename='WARD';
```

Display all managers whose salary is more than 2500:

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

Display all managers who joined after may 1981:

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

Online Shopping:
searching for products

searching for DELL company and APPLE company Laptops:

Products Table => cname Column

```
WHERE cname='APLLE' OR cname='DELL'
```

searching for DELL company laptops, price should be b/w 50000 and 70000:

```
WHERE cname='DELL' AND  (price>=50000 AND price<=70000)
```

NOT:

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

Truth Table:

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

Display all emp records except managers records:

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

```
job
---------
CLERK
MANAGER
SALESMAN
MANAGER
ANALYST
```

IN:

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

Syntax:

```
WHERE <column> IN(<value_list>)
```

If column value is IN list then condition is TRUE
If column value not IN the list then condition is FALSE

Examples on IN operator:

Display all managers and clerks records:

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

WHERE job='MANAGER' OR job='CLERK'	WHERE job IN('CLERK','MANAGER')

Display the emp records whose empnos are 7521, 7698, 7900:

```
SELECT *
FROM emp
WHERE empno IN(7521,7698,7900);
```

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

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

Display all emp records except managers and clerks:

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

job
----------
MANAGER
ANALYST
CLERK
SALESMAN
```

If job value NOT IN the list then condition is TRUE
If job value IN the list then the condition is FALSE

Display the emp records who are no working in 10 and 30 depts:

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

BETWEEN AND:

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

Syntax:

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

If column value falls under specified range then condition is TRUE

Examples on BETWEEN AND:

Display the emp records whose salary is between 2450 and 3000:

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

Display the emp records who joined in 1982:

```
SELECT ename,hiredate
FROM emp
WHERE hiredate BETWEEN '1-JAN-1982' AND '31-DEC_1982';
```

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

What is the Output?

A. displays salaries b/w 2450 and 3000
B. ERROR

C. No rows Selected
D. None

Answer: C

Display the emp records who are not joined in 1981:

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

LIKE:

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

ORACLE SQL provides 2 wildcard characters. They are:

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

Syntax:

```
WHERE <column> LIKE <text_pattern>
```

If column value is matched with text pattern then condition is TRUE.

Display the emp records whose name is started with S:

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

Display the emp records whose names are ended with S:

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

Display the emp records whose names are having 4 chars:

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

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

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

Display the emp records who joined in DECEMBER month:

```
SELECT ename,hiredate
```

```
FROM emp
WHERE hiredate LIKE '%DEC%';
```

Display the emp records who are earning 3 digit salary:

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

Display the emp records whose name's 3rd char is M:

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

Display the emp records whose name is having M char:

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

Display the emp records whose name is started and ended with S:

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

Display the emp names which are having _:

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

(or)

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

Display the emp records whose name is having %:

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

NULL:

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

E.F.Codd Rule:

- NULL means empty or blank.
- NULL is not equals to 0 or space.
- When we don't know the value or when we are unable to insert the value, we insert NULL.
- If NULL is participated in operation then result will be NULL.
  100+200 => 300
  100+200+NULL => NULL
- For NULL comparison, we cannot use =.
  For NULL comparison always use IS NULL operator.

E.F.Codd
Rule:
null is not equals to 0 or space

ORACLE 2,3,4,5,6 => VARCHAR()
VARCHAR data type was treating space and null as same.

ORACLE 7 => VARCHAR2()

NULL can be inserted in 2 ways. They are:

- Direct way : using NULL keyword
- Indirect way: by inserting limited column values

Example:

T1

F1	F2
1001	ABC

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

1001	ABC

```
INSERT INTO t1 VALUES(1001,'ABC');
```

1002	

Direct way:

```
INSERT INTO t1 VALUES(1002,NULL);
```

1003	

Indirect way:

```
INSERT INTO t1(f1) VALUES(1003);
```

IS NULL:

- For NULL comparison we cannot use =
- For NULL comparison we must use IS NULL operator.
- IS NULL operator is used to compare column value with NULL.

Syntax:

```
WHERE <column> IS null
```

Examples:

Display the emp records who are not getting commission:

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

Output:
no rows selected **null=null FALSE**

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

Display the emp records who are getting commission:

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

Display the emp records whose name is not started with S:

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

Concatenation Operator:

- **Symbol: ||**
- **concatenate => combine**
- **used to combine 2 strings**

Display output as following:
SMITH earns 800
ALLEN earns 1600

```
SELECT ename  || ' earns ' || sal FROM emp;
```

SMITH earns 800

TCL commands

Thursday, August 17, 2023 6:17 PM

TCL:

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

- **Transaction:**
  Transaction is a series of actions [SQL commands]

  Examples:
  Withdraw, Deposit, Check Balance, Placing Order

Withdraw Transaction

machine reads account details
enter PIN: 1234 [SELECT]
displays menu
withdraw
enter amount: 10000 [SELECT]
machine gives money
UPDATE the balance [UPDATE]

BANK DB

Accounts

Acno	PIN	Balance
1001	1234	~~50000~~ 40000

Rule:
Transaction must be successfully finished or cancelled

Note:
If transaction is successful, to save it use COMMIT
If transaction is unsuccessful, to cancel it use ROLLBACK

ORACLE SQL provides 3 TCL commands. They are:

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

COMMIT [save]:

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

Syntax:

```
COMMIT;
```

ROLLBACK [undo]:

- **ROLLBACK command is used to cancel the uncommitted actions.**
- **After COMMIT we cannot use ROLLBACK.**

Syntax:

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

Example on Commit and Rollback:

T1

F1
1001
1002
1003

```
CREATE TABLE t1
(
f1 NUMBER(4)
);
```

Output:
Table created.

```
INSERT INTO t1 VALUES(1001);        -- inserts in INSTANCE [RAM]
INSERT INTO t1 VALUES(1002);        -- inserts in INSTANCE [RAM]
COMMIT;                             --saves in DB

INSERT INTO t1 VALUES(1003);        -- inserts in INSTANCE [RAM]
INSERT INTO t1 VALUES(1004);        -- inserts in INSTANCE [RAM]

SELECT * FROM t1;
Output:
1001
1002
1003
1004

ROLLBACK;                           -- 2 actions will be cancelled

SELECT * FROM t1;
Output:
1001
1002
```

SAVEPOINT:

- **SAVEPOINT is used to margin for ROLLBACK.**
- **Using SAVEPOINT, we can cancel part of the transaction.**

Syntax:

```
SAVEPOINT <savepoint_name>;
```

Example:

```
CREATE TABLE t2
INSERT => 1001
INSERT => 1002
INSERT => 1003
```

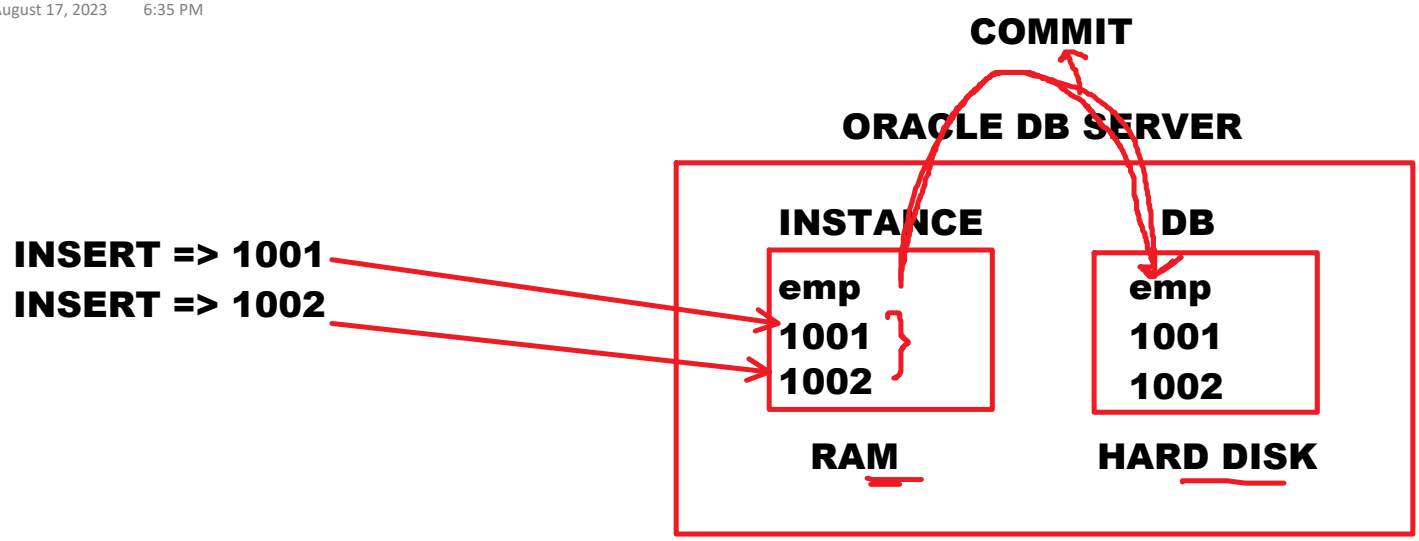
INSERT => 1004
INSERT => 1005
INSERT => 1006

ROLLBACK; --6actions cancelled

Example:
CREATE TABLE t3
INSERT => 1001
INSERT => 1002
SAVEPOINT p1;
INSERT => 1003
INSERT => 1004
SAVEPOINT p2;
INSERT => 1005
INSERT => 1006

ROLLBACK TO p2; --2 actions cancelled

to save the actions	**COMMIT**
to cancel actions	**ROLLBACK**
to set margin for ROLLBACK	**SAVEPOINT**



SERVER = INSTANCE + DB

DDL commands

Thursday, August 17, 2023 7:13 PM

ALTER:

- **ALTER => change**
- **ALTER command is used to change structure of the table**

- **Using ALTER command we can:**
  - **Add the columns => ADD**
  - **Rename the columns => RENAME COLUMN**
  - **Drop the columns => DROP**
  - **Modify the field sizes => MODIFY**
  - **Modify the data types => MODIFY**

Syntax:

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

Example on ALTER commands:

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

DESC student;
Output:
sid       NUMBER(4)
sname VARCHAR2(10)
```

Adding a new Column [m1]:

```
ALTER TABLE student ADD m1 NUMBER(3);
Output:
Table Altered

DESC student;
Output:
```

```
sid      NUMBER(4)
sname VARCHAR2(10)
m1       NUMBER(3)
```

Adding multiple columns [m2, m3]:

```
ALTER TABLE student ADD(m2 NUMBER(3), m3 NUMBER(3));
Output:
Table Altered

DESC student;
Output:
sid      NUMBER(4)
sname VARCHAR2(10)
m1       NUMBER(3)
m2       NUMBER(3)
m3       NUMBER(3)
```

Renaming a column [m3 TO maths]:

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

DESC student;
Output:
sid      NUMBER(4)
sname VARCHAR2(10)
m1       NUMBER(3)
m2       NUMBER(3)
maths  NUMBER(3)
```

Dropping a Column [maths]:

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

DESC student;
Output:
sid      NUMBER(4)
```

sname VARCHAR2(10)
m1 NUMBER(3)
m2 NUMBER(3)

Dropping multiple columns [m1, m2]:

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

DESC student;
Output:
sid      NUMBER(4)
sname VARCHAR2(10)
```

Modifying filed size [sname varchar2(10) => increase size from 10 to 20]:

```
ALTER TABLE student
MODIFY sname VARCHAR2(20);

Output:
Table Altered

DESC student;
Output:
sid      NUMBER(4)
sname VARCHAR2(20)
```

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

```
sname
-----------
KIRAN   => 5
RAJU    =>  4
SAI     => 3
RAMESH => 6
```

```
ALTER TABLE student
MODIFY sname VARCHAR2(4);

ERROR:
```

We can decrease up to 6 only
We cannot decrease less than 6

Modifying data type sid NUMBER(4) => [convert number to char]:

STUDENT

sid NUMBER(4) => CHAR(8)	sname
HYD_1001	
DLH_1002	
CHN_1003	

ALTER TABLE student
MODIFY sid CHAR(8);
Output:
Table Altered.

Note:
To modify the data type column must be empty

DESC student;
Output:
sid CHAR(8)
sname VARCHAR2(20)

DROP, FLASHBACK and PURGE

Friday, August 18, 2023 6:13 PM

- In ORACL 10g version recyclebin concept is added
- FLASHBACK and PURGE are related to recyclebin.

- FLASHBACK and PURGE commands introduced in ORACLE 10g version.

DROP
FLASHBACK
PURGE

DROP:

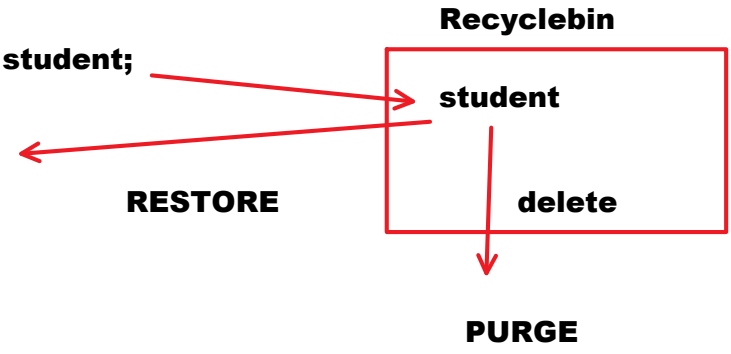
- DROP command is used to delete the table.
- When table is dropped it goes to recyclebin.

Syntax:

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

Example:

```
DROP TABLE student;
```



FLASHBACK:

- It is used to restore the dropped table

Syntax:

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

Example:

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

PURGE:

- It is used to delete the table from recyclebin.
- With this, table will be deleted permanently.

Syntax:

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

Example:

```
PURGE TABLE student;
```

To see recyclebin:

```
show recyclebin
```

Example on DROP, FLASHBACK and PURGE:

```
CREATE TABLE t1
(
f1 NUMBER(4)
);

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

T1
F1
1
2

Drop t1 table:

```
DROP TABLE t1;
--it goes to recyclebin

SELECT * FROM t1;
--ERROR: table does not exist

show recyclebin
Output:
original_name       recyclebin_name
-------------------------   -------------------------
T1                          --
```

Restore t1 Table:

```
FLASHBACK TABLE t1 TO BEFORE DROP;
-- restores t1 table

SELECT * FROM t1;
--displays table data
```

Deleting From Recyclebin:

```
DROP TABLE t1;

PURGE TABLE t1;
--t1 table deleted permanently
```

Deleting a table permanent:

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

```
DROP TABLE student PURGE;

--student table will be dropped permanently
--it will not be placed in recyclebin
```

Emptying recyclebin:

```
PURGE recyclebin;
```

Example:

```
drop table t1;
drop table t2;
drop table t3;
.
.
drop table t10;
```

```
recyclebin
t1
t2
t3
.
.
t10
```

```
PURGE TABLE t1;
PURGE TABLE t2;
.
```

```
PURGE RECYCLEBIN;
--empties recyclebin
```

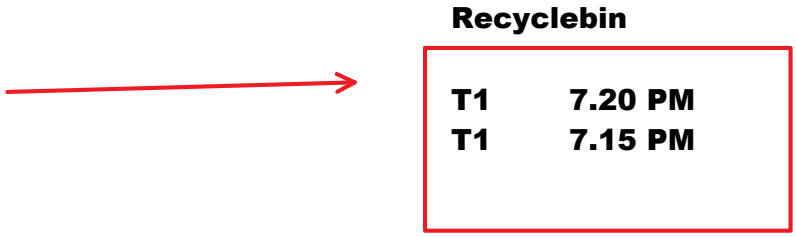
.

PURGE TABLE t10;

CASE-1:

CREATE TABLE t1
DROP TABLE t1;

CREATE TABLE t1
DROP TABLE t1;



FLASHBACK TABLE t1
TO BEFORE DROP;
--recent one will be restored

restore the table dropped at 7:15 [older t1]:

FLASHBACK TABLE <recyclebin_name>
TO BEFORE DROP;

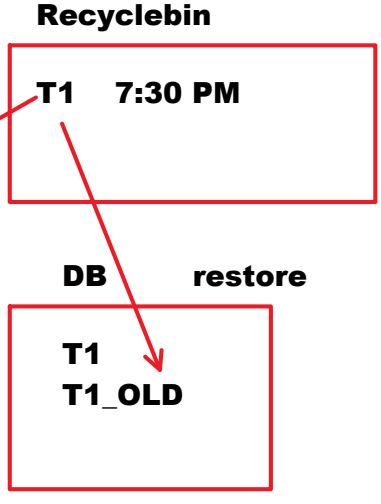
FLASHBACK TABLE
"BIN$4UpKoXO5RCmmugMPc28n+g==$0"
TO BEFORE DROP;

CASE-2:

CREATE TABLE t1
DROP TABLE t1

CREATE TABLE t1

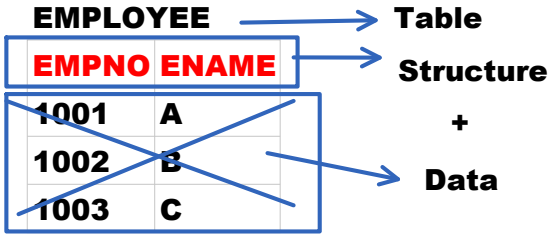
FLASHBACK TABLE t1
TO BEFORE DROP;
Output:
ERROR: name already
used by existing object



```
FLASHBACK TABLE t1
TO BEFORE DROP
RENAME TO t1_old;
```

TRUNCATE:

TABLE = STRUCTURE + DATA



To delete entire table data use TRUNCATE

- **TRUNCATE command is used to delete entire table data. It means, it deletes all records from table.**
- **To delete all records from table with good performance, use TRUNCATE command.**

Syntax:

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

Example:

```
TRUNCATE TABLE t1;
```

DROP	• used to delete the table • it deletes table data + structure • it can be flashed back

TRUNCATE	• **used to delete all records** • **it does not delete table structure** • **it cannot be flashed back**

RENAME:

- **is used to change table name**

Syntax:

RENAME <old_name> TO <new_name>;

Example:

RENAME employee TO emp;

CREATE => to create the table
ALTER => to change structure of table

DROP => to delete the table
FLASHBACK => to restore the table
PURGE => to delete from recyclebin

TRUNCATE => to delete all records
RENAME => to rename the table

DML

Saturday, August 19, 2023 6:36 PM

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 <field_name> = <value> [, <field_name> = <value> , ...]
[WHERE <condition>];
```

- **modifying single value of single record:**

Set sal as 4000 to the employee whose empno is 7521:

```
UPDATE emp
SET sal=4000
WHERE empno=7521;
```

Output:

1 row updated

- **modify multiple values of single record:**

Set job as manager, salary as 6000 to an employee whose empno is 7369:

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

- modify specific group of records:

  Increase 2000 rupees salary to all managers:

```
UPDATE emp
SET sal=sal+2000
WHERE job='MANAGER';
```

modify all records:

Increase 10% on sal to all emps:

10/100 = 0.1

```
UPDATE emp
SET sal=sal+sal*0.1;
```

Examples on UPDATE command:

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

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

Set comm as 700 to the emps who are not getting commission:

```
UPDATE emp
SET comm=700
WHERE comm is null;
```

Set comm as null to all managers:

```
UPDATE emp
SET comm=null
WHERE job='MANAGER';
```

Transfer all 10th dept emps to 20th dept:

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

Increase 10% on salary to the employees who are having more than 41years experience:

```
UPDATE emp
SET sal=sal+sal*0.1
WHERE TRUNC((sysdate-hiredate)/365)>41;
```

DELETE:

- **DELETE command is used to delete the records**

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

Syntax:

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

Deleting single record:

Delete an emp record whose empno is 7900:

```
DELETE FROM emp
WHERE empno=7900;
Output:
1 row deleted.
```

Deleting specific group of records:

Delete all managers records:

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

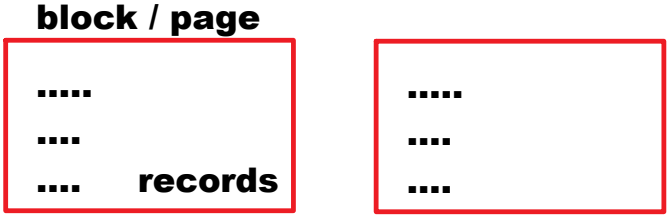
Deleting all records:

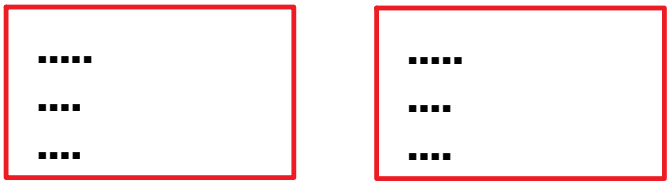
```
DELETE FROM emp;
(or)
DELETE emp;
```

Using DELETE command we can delete all records.
Using TRUNCATE command we can delete all records.

What are the differences between DELETE & TRUNCATE?

TRUNCATE	DELETE
• it is DDL command	• it is DML command
• it deletes all records. it cannot delete sing record or a specific group of records.	• it can delete all records, single record or a specific group of records
• 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 page by page [page = block]	• It deletes row by row





Note:

- **All DDL commands are auto committed by default**
- **All DML commands are not auto committed by default**

When DDL command is executed, implicitly COMMIT command will be executed.

CREATE = CREATE + COMMIT
ALTER = ALTER + COMMIT

CREATE TABLE t1 => CREATE + COMMIT => COMMITTED
INSERT
INSERT
INSERT
INSERT
ROLLBACK => 4 actions will be cancelled

CREATE TABLE t1 => CREATE + COMMIT => COMMITTED
INSERT
INSERT

CREATE TABLE t2 => CREATE + COMMIT => COMMITTED
INSERT
INSERT
ROLLBACK => 2 actions will be cancelled

SQL

DDL	DRL	TCL	DML	DCL
create alter drop flashback purge truncate rename	select	commit rollback savepoint	insert update delete insert all merge	grant revoke

Copying table & Copying records:

Copying Table:

Syntax:

```
CREATE TABLE <table_name>
AS
<SELECT query>;
```

- **Copying table means, creating a new table from existing table.**
- **To copy the table we use CREATE command only.**
- **With SELECT query result a new table will be created.**

Examples:

create exact copy of "emp" table with the name "emp1":

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

create a new table with the name "emp2", with all managers records of emp table, with 4 columns empno,ename,job,sal:

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

Copying table structure:

- **To copy table structure from existing table, write select query with FALSE condition.**
- **Because of always condition is FALSE, rows will not be selected.**

Create a new table from existing table emp with the name "emp3", with 4 columns empno,ename,job,sal without records. [copying table structure]

```
CREATE TABLE emp3
AS
SELECT empno,ename,job,sal
FROM emp
WHERE 1=2;
```

Copying records:

Syntax:

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

- For copying records we use INSERT command.
- In this, SELECT query result will be inserted into existing table.

Example:

Table emp => existing table

8 columns
14 rows

copy

Table emp4

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

copy emp table all rows to emp4:

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

```
INSERT INTO emp4
SELECT empno,ename,job,sal
FROM emp;
```

INSERT ALL

Monday, August 21, 2023 7:09 PM

INSERT ALL:

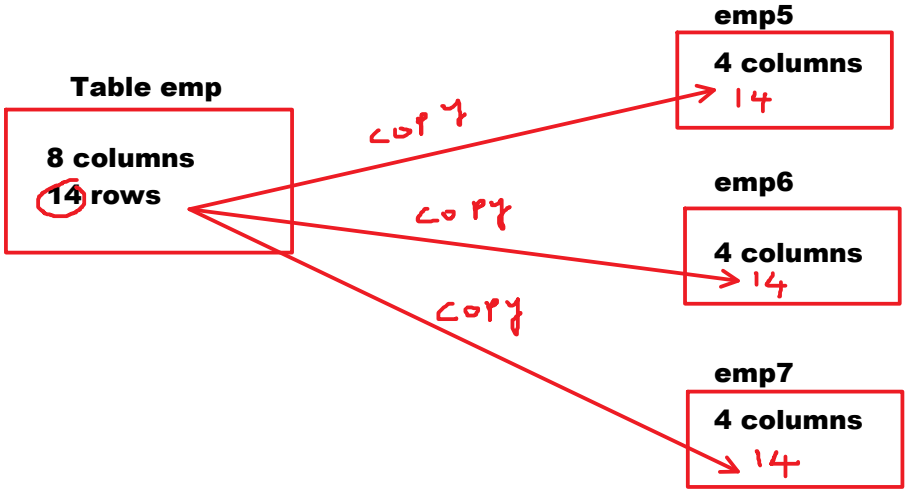
- **INSERT ALL command introduced in ORACLE 9i version.**
- **It is used to copy one table data to multiple tables.**
- **It avoids of writing many INSERT commands.**
- **It can be used in 2 ways. They are:**
  - **Unconditional INSERT ALL**
  - **Conditional INSERT ALL**

Unconditional INSERT ALL:

Syntax:

```
INSERT ALL
INTO <table_name>[(<column_list>)] VALUES(<value_list>)
INTO <table_name>[(<column_list>)] VALUES(<value_list>)
.
.
<SELECT query>;
```

Example on Unconditional INSERT ALL:



Create emp5, emp6, emp7 tables with 4 columns empno,ename,job,sal and without any records. create these tables from existing table emp:

CREATE TABLE emp5

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


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


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

Copy emp table all records to emp5, emp6 and emp7:

```
INSERT ALL
INTO emp5 VALUES(empno,ename,job,sal)
INTO emp6 VALUES(empno,ename,job,sal)
INTO emp7 VALUES(empno,ename,job,sal)
SELECT empno,ename,job,sal FROM emp;
```

Output:

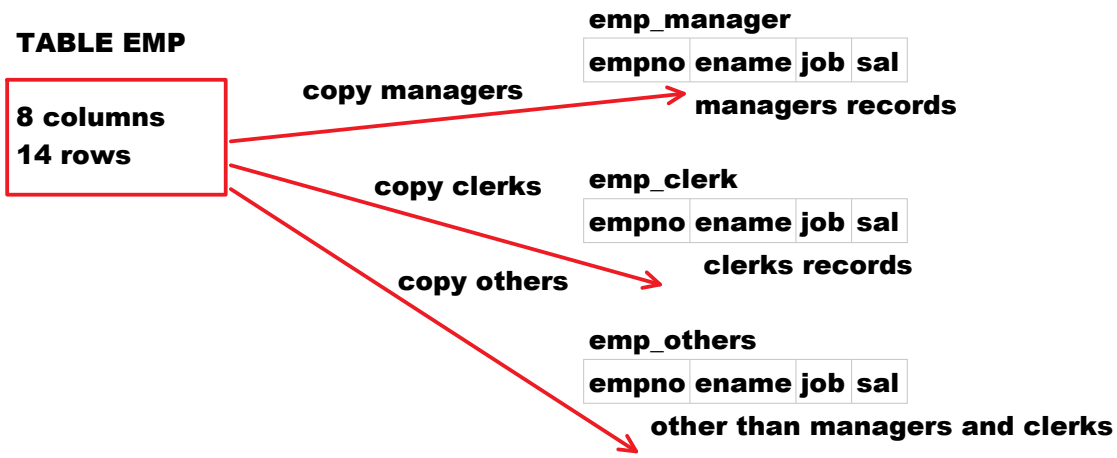
42 rows created.

Conditional INSERT ALL:

```
INSERT ALL
WHEN <condition-1> THEN
   INTO <table_name>(<column_list>) VALUES(<value_list>)
WHEN <condition-2> THEN
   INTO <table_name>(<column_list>) VALUES(<value_list>)
.
.
[ELSE
INTO <table_name>(<column_list>) VALUES(<value_list>)]
<SELECT query>;
```

```
if(condn1)
{
}
if(condn2)
{
}
.
.
```

Example on Conditional INSERT ALL:



Create emp_manager, emp_clerk and emp_others tables from existing table emp with 4 columns empno, ename, job, sal without records:

```
CREATE TABLE emp_manager
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;



INSERT ALL
WHEN job='MANAGER' THEN
INTO emp_manager 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;
```

```
if(condn1)
{
}
else if(condn2)
{
}
else if(condn3)
{
}
```

```
INSERT FIRST
WHEN job='MANAGER' THEN
INTO emp_manager 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;
```

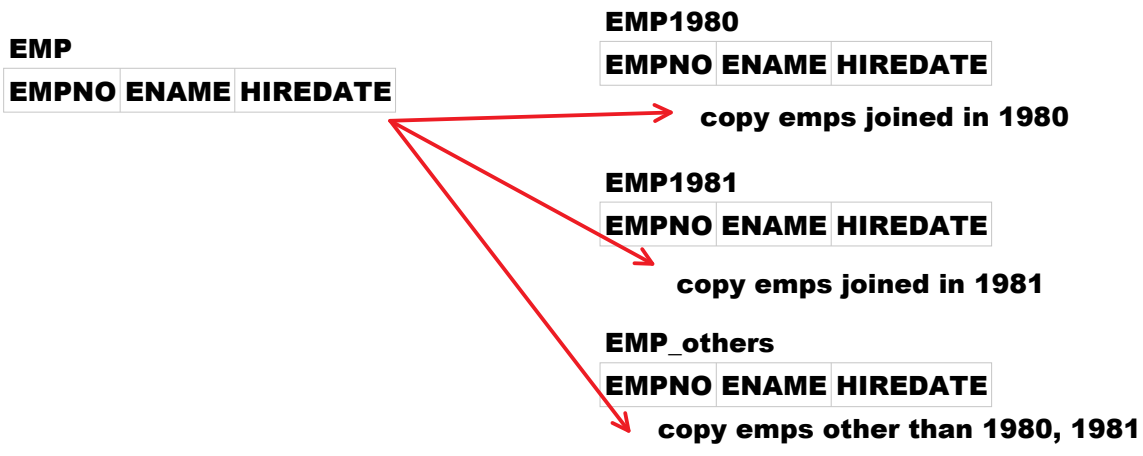
Note:
In "INSERT ALL", "WHEN .. THEN" acts like "if" control structure in C/Java.

In "INSERT FIRST", "WHEN .. THEN" acts like "if else if" control structure in C/Java.

In INSERT ALL,
even if first condition is TRUE or FALSE it checks next condition.

In INSERT FIRST,
if first condition is TRUE, it will not check remaining conditions
if first condition is FALSE, then only it checks 2nd condition

Assignment:



MERGE

Wednesday, August 23, 2023 6:48 PM

S.cid = T.cid

BRANCH OFFICE

CUSTOMER1 S

cid	cname	ccity
1	A	HYD BENGALORE
2	B	DELHI
3	C	CHENNAI
4	D	PUNE
5	E	KOLKATA

matched
UPDATE

not matched
INSERT

HEAD OFFICE

CUSTOMER2 T - REPLICA [duplicate copy]

cid	cname	ccity
1	A	HYD BENGALORE
2	B	DELHI
3	C	CHENNAI
4		
5		

MERGE:

- **MERGE command introduced in Oracle 9i version.**
- **It is used to deal with REPLICAS [duplicate copies].**
- **Before ORACLE 9i version, we were writing a separate PL/SQL program to deal with replicas.**
- **MERGE = UPDATE + INSERT**
- **MERGE is a combination of UPDATE and INSERT commands**
- **MERGE command can be also called as "UPSERT" command**
- **MERGE command is used to apply one table changes to its REPLICA [duplicate copy]**

Syntax of MERGE command:

```
MERGE INTO <target_table_name> <table_alias>
USING <source_table_name> <table_alias>
ON(<condition>)
WHEN matched THEN
   UPDATE query
WHEN not matched THEN
   INSERT query;
```

Example on MERGE command:

CUSTOMER1

cid	cname	ccity
1	A	HYD
2	B	DELHI
3	C	CHENNAI

```
CREATE TABLE customer1
(
cid NUMBER(4),
cname VARCHAr2(10),
ccity VARCHAr2(10)
);

INSERT INTO customer1 VALUES(1,'A','HYD');
INSERT INTO customer1 VALUES(2,'B','DELHI');
INSERT INTO customer1 VALUES(3,'C','CHENNAI');
COMMIT;

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

4	D	PUNE
5	E	KOLKATA

```
INSERT INTO customer1 VALUES(4,'D','PUNE');
INSERT INTO customer1 VALUES(5,'E','KOLKATA');
COMMIT;
```

1	A	HYD BENGALORE

```
UPDATE customer1
SET ccity='BENGALORE'
WHERE cid=1;

COMMIT;
```

CUSTOMER1 S

cid	cname	ccity
1	A	~~HYD~~ BENGALORE
2	B	DELHI
3	C	CHENNAI
4	D	PUNE
5	E	KOLKATA

CUSTOMER2 T

cid	cname	ccity
1	A	HYD
2	B	DELHI
3	C	CHENNAI
s.cid	s.cname	s.ccity

Apply customer1 table changes to its replica customer2:

```
MERGE INTO customer2 T
USING customer1 S
ON(S.cid = T.cid)
WHEN matched THEN
UPDATE SET T.cname=S.cname, T.ccity=S.ccity
WHEN not matched THEN
INSERT VALUES(s.cid, s.cname, s.ccity);
```

Output:
5 rows merged

DCL

Thursday, August 24, 2023 6:14 PM

DCL:

- **DCL => Data Control Language.**
- **It deals with data accessibility.**

ORACLE SQL provides 2 DCL commands. They are:

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

GRANT:

- **GRANT command is used to give permission to other users on DB objects [tables, views].**

  Syntax:

```
GRANT <privileges_list>
ON <table_name>
TO <user_list>;
```

  Examples:

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

GRANT insert,upddate,delete ON emp TO c##userA;

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

REVOKE:

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

  Syntax:

```
REVOKE <privileges_list>
ON <table_name>
FROM <user_list>;
```

Examples:

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

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

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

Example on GRANT & REVOKE:

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

Login as DBA:
username: system
password: nareshit

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

Output:
User created

```
GRANT create session, create table, unlimited tablespace
TO c##userA;
```

Output:
Grant succeeded

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

Output:
User created

```
GRANT create session, create table, unlimited tablespace
TO c##userB;
```

Output:
Grant succeeded

Note: press windows+right arrow to arrange windows

c##userA	c##userB

T1

F1	F2
1	A
2	B

```
CREATE TABLE t1
(
f1 NUMBER(4),
f2 VARCHAR2(10)
);

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

```
SELECT * FROM c##userA.t1;
Output:
ERROR: table does not exist
```

```
GRANT select
ON t1
TO c##userB;
Output:
Grant succeeded
```

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

F1	F2
1	A
2	B

```
INSERT INTO c##userA.t1
VALUES(3,'C');
Output:
ERROR: insufficient privileges
```

ERROR: insufficient privileges

UPDATE c##userA.t1
SET f1='SAI'
WHERE f1=1;
Output:
ERROR: insufficient privileges

DELETE FROM c##userA.t1
WHERE f1=2;
Output:
ERROR: insufficient privileges

GRANT insert,update,delete
ON t1
TO c##userB;
Output:
Grant succeeded

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

SELECT * FROM t1;
Output:

F1	F2
1	A
2	B

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

F1	F2
1	A
2	B
3	C

COMMIT;

SELECT * FROM t1;
Output:

F1	F2
1	A
2	B
3	C

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

1 row updated

DELETE FROM c##userA.t1
WHERE f1=2;
Output:
1 row deleted

COMMIT;

GRANT all
ON t1
TO c##userB;
Output:
Grant succeeded

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

INSERT => ERROR
UPDATE => ERROR
DELETE => ERROR

REVOKE all
ON t1
FROM c##userB;

WITH GRANT OPTION:
WITH GRANT OPTION clause is used to

allow the GRANTEE to give permission to other users on GRANTOR's DB Object.

GRANTOR	**c##userA** **GRANT select** **ON t1** **TO c##userB** **WITH GRANT OPTION;**
GRANTEE	**c##userB** **GRANT all** **ON c##userA.t1** **TO c##batch6pm;** **Output:** **Grant succeeded**

c##userB has SELECT permission only

all	**SELECT**

c##userA:

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

//implicitly c##batch6pm permissions will be cancelled
```

user_tab_privs_made:

- **it is a system table / built-in table**
- **it maintains list of privileges [permissions] made by the user**

user_tab_privs_recd:

- **it is a system table / built-in table**
- **It maintains list of privileges received by the user**

to see list of permissions received by user:

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

to see list of permissions made by user:

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

DUAL

Friday, August 25, 2023 6:36 PM

DUAL:

- **DUAL is a system table / built-in table / readymade table**
- **DUAL table created in "SYS" Schema [user]**
- **It has 1 column, 1 row**

DUAL

DUMMY VARCHAR2(1)
X

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

  Example:

  SELECT 100+200 FROM dual;
  Output:
  300

  DUAL table has 1 row so, displays one 300

  SELECT 100+200 FROM emp;
  Output:
  300
  300
  .
  .
  300 14 300s

  emp table has 14 rows. for every row 300 value returned once

  Till ORACLE 21C:
  SELECT lower('RAJU') FROM dual;

Till ORACLE 21C FROM clause is mandatory

ORACLE 23C [LINUX]:
From ORACLE 23C version onwards, using FROM clause is optional.

SELECT lower('RAJU')
Output:
raju

Built-In Functions

Friday, August 25, 2023 6:18 PM

Built-In Functions:

- **Built-In Functions can be also called also predefined functions / readymade functions.**
- **To make our tasks easier ORACLE SQL provides built-in functions.**
- **function => action / task. Every function performs particular action.**

ORACLE SQL built-in functions can be categorized as following:

- **String Functions**
- **Conversion Functions**
- **Date Functions**
- **Aggregate Functions [Group Functions]**
- **Number Functions**
- **Analytic Functions / Window Functions**
- **Miscellaneous Functions**

String Functions:

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

lower():

- **it is used to convert the string to lower case**

Syntax:

lower(<string>)

Examples:

lower('RAJU')	**'raju'**
lower('RAJ KUMAR')	**'raj kumar'**

SELECT lower('RAJU') FROM dual;
Output:
raju

upper():
It is used to convert the string to upper case.

Syntax:
upper(<string>)

Example:

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

initcap() [initial capital letter]:
- **It is used to get every word's starting letter as capital.**

Syntax:
initcap(<string>)

Examples:

initcap('RAJU')	Raju
initcap('RAJ KUMAR VARMA')	Raj Kumar Varma

concat() [concatenate / combine]:
used to combine 2 strings

Syntax:
concat(<string1>, string2>)

Examples:

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

```
SELECT 'RAJ' || 'KUMAR' || 'VARMA' FROM dual;
Output:
RAJKUMARVARMA

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

length():
- **used to find string length**
- **string length = no of chars in string**

Syntax:
length(<string>)

Examples:

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

Example queries:

Display all emp records. display emp names in lower case:

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

Modify all emp names to initcap case:

```
UPDATE emp
SET ename = initcap(ename);
Output:
14 rows updated
```

Display the emp record whose name is BLAKE when we don't know exact case:

```
SELECT *
FROM emp
WHERE ename = 'blake';
Output:
no rows selected

BLAKE = blake  => F



SELECT *
FROM emp
WHERE lower(ename) = 'blake';
Output:
displays BLAKE record

SELECT *
FROM emp
WHERE upper(ename) = 'BLAKE';
Output:
displays BLAKE record


SELECT *
FROM emp
WHERE initcap(ename) = 'Blake';
Output:
displays BLAKE record
```

Display emp records whose names are having 4 chars:

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

(or)
   SELECT *
   FROM emp
```

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

Display emp records whose names are having 14 chars:

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

Display the emp records whose name has more than 10 chars:

```
SELECT *
FROM emp
WHERE length(ename)>10;
```

Assignment:

PLAYER

PID	FNAME	LNAME
1001	VIRAT	KOHLI
1002	ROHIT	SHARMA

```
PNAME
-----------
Virat Kohli
Rohit Sharma
```

add a column => PNAME
concatenate fname and lname and place it in PNAME column in initcap case
drop fname and lname

- **Display the player names which are having 12 chars**
- **Display the player names which are having less than 12 chars**

Reverse():
used to get reverse string

Syntax:

```
Reverse(<string>)
```

Example:

Reverse('RAJU')	UJAR

LPAD() & RPAD():

- **PAD => Fill**

LPAD():

- **used to fill specified char set at left side**

Syntax:

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

3rd argument default char => space

RPAD():

- **used to fill specified char set at right side**

Syntax:

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

3rd argument default char => space

no of chars to be filled = max size - string length

Examples:

LPAD('RAJU',10,'*')	******RAJU
RPAD('RAJU',10,'*')	RAJU******
LPAD('RAJU',8)	4spacesRAJU
RPAD('RAJU',8)	RAJU4spaces
LPAD('A',6,'A')	AAAAAA
LPAD('X',6,'X')	XXXXXX

Substr():

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

Syntax:

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

Examples:

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

Substr('RAVI TEJA',6)	TEJA
Substr('RAVI TEJA',6,3)	TEJ
Substr('RAVI TEJA',1,4)	RAVI
Substr('RAVI TEJA',3,4)	VI T

2nd argument can be negative

if 2nd argument is,

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

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

Substr('RAVI TEJA',-4)	TEJA
Substr('RAVI TEJA',-4,3)	TEJ

Substr('RAVI TEJA',-9,4)	RAVI

Generate email id to all emps by taking
emp name first 3 chars, empno last 3 chars as username
for the domain 'tcs.com':

EMPNO	ENAME	MAIL_ID
7369	SMITH	SMI369@tcs.com
7499	ALLEN	ALL499@tcs.com

Add a new column Mail ID:

```
ALTER TABLE emp ADD mail_id VARCHAR2(20);
```

generate mail ids to all emps:

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

acno: 1234561234

amount debited from account number XXXXXX1234

```
SELECT
'amount debited from account number ' ||  LPAD('X',6,'X') ||
Substr('1234561234',-4,4)
FROM dual;
```

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

```
ename
----------
SMITH
ALLEN
SCOTT
```

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

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

```
ename
-----------
SMITH
WARD
ALLEN
FORD
```

```
SELECT ename,sal
FROM emp
WHERE substr(ename,-2,2) = 'RD';
```

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

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

```
ename
----------
SMITH    =>   S = H   FALSE
DAVID    =>   D = D   TRUE
SYMONDS  =>   S = S   TRUE
```

Display the emp names whose name started with Vowel:

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

Instr():

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

Syntax:

```
Instr(<string>, <sub string> [, <search_position>, <occurrence>])
```

3rd arg => search position default value is 1
4th arg => occurrence default value is 1

Examples:

Instr('RAVI TEJA','TEJA')	6
Instr('RAVI TEJA','SAI')	0
Instr('RAVI TEJA RAVI TEJA','TEJA',1,2)	16
Instr('RAVI TEJA RAVI TEJA','RAVI')	1
Instr('RAVI TEJA RAVI TEJA','RAVI',3)	11

1 2 3 4 5 6 7 8 9 10 11 12 13 14

R	A	V	I		T	E	J	A		R	A	V	I
-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Instr('RAVI TEJA RAVI','RAVI',-1)	11
Instr('RAVI TEJA RAVI','RAVI',-1,2)	1
Instr('RAVI TEJA RAVI','TEJA',-3,2)	0

Example:
Display the emp names whose names are having AM chars:

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

Assignment:

PNAME	
VIRAT KOHLI	
SACHIN TENDULKAR	

FNAME	LNAME
VIRAT	KOHLI
SACHIN	TENDULKAR

```
PNAME
-----------------
RAJ KUMAR VARMA
SACHIN RAMESH TENDULKAR
```

FNAME	MNAME	LNAME
RAJ	KUMAR	VARMA

Replace() & Translate():

Replace():

- **is used to replace search string with replace string**

Syntax:

Replace(<string>, <serach_string>, <replace_string>)

Examples:

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

Translate():

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

Syntax:

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

Examples:

Translate('HELLO WELCOME','LO','ab')	HEaab WEaCbME

L	a
O	b

Replace('HELLO WELCOME','LO','ab')	HELab WELCOME

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

Translate('abcabcaabbccabc','abc','XYZ')	XYZXYZXXYYZZXYZ

a	X
b	Y
c	Z

Replace()	replaces the strings
Translate()	replaces the chars

Ltrim(), Rtrim() & Trim():

- **Trim => remove**

Ltrim():

- **used to remove unwanted chars from left side**

Syntax:

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

2nd arg default char => space

Rtrim():

- **used to remove unwanted chars from right side**

Syntax:

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

2nd arg default char => space

Trim():

- **can be used to remove unwanted char from left side or right side or both sides.**

Syntax:

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

Examples:

Ltrim('***RAJU***', '*')	RAJU***
Rtrim('***RAJU***', '*')	***RAJU
Ltrim(' RAJU ')	RAJu3spaces
Rtrim(' RAJU ')	3spacesRAJU

Trim(LEADING '@' FROM '@@@RAJU@@@')	RAJU@@@
Trim(TRAILING '@' FROM '@@@RAJU@@@')	@@@RAJU
Trim(BOTH '@' FROM '@@@RAJU@@@')	RAJU
Trim(' RAJU ') default side=> both default char => space	RAJU

Conversion Functions:

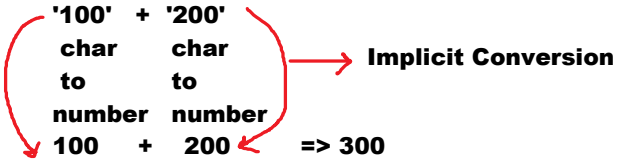
ORACLE SQL supports to 2 types of conversions. They are:

- **Implicit Conversion**
- **Explicit Conversion**

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

Example:

```
SELECT '100' + '200' FROM dual;
Output:
300
```



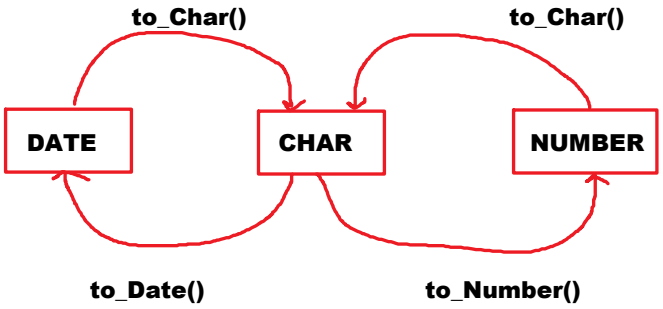
Explicit Conversion:
If conversion is done using built-in function then it is called "Explicit Conversion".

For Explicit Conversion, we use following functions:

- **to_Char()**
- **to_Date()**
- **to_Number()**

to_Char() to_Char()

- to_Number()



Note:

- **Implicit Conversion degrades the performance.**
- **Don't depend on Implicit Conversion. Always do explicit conversion.**

to_Char() [Date to Char]:

- **to_Char() function can be used to convert date to char [string]**
- **Using to_char(), we can change date formats from ORACLE DATE FORMAT TO another like IND date format [DD/MM/YYYY], US date format [MM/DD/YYYY]**
- **It can be to get part of the date like year, month, day, weekday, ...etc.**

Syntax:

to_Char(<date>, <format>)

FORMAT	PURPOSE	EXAMPLE sysdate => 28-AUG-23	OUTPUT
YYYY	year 4 digits	to_char(sysdate,'YYYY')	2023
YY	year last 2 digits	to_char(sysdate,'YY')	23
YEAR / year	year in words	to_char(sysdate,'YEAR') to_char(sysdate,'year')	TWENTY TWENTY THREE twenty twenty three
MM	month number	to_char(sysdate,'MM')	08
MON / mon	short month name	to_char(sysdate,'MON') to_char(sysdate,'mon')	AUG aug
MONTH / month	full month name	to_char(sysdate,'MONTH') to_char(sysdate,'month')	AUGUST august
DD	day num in month	to_char(sysdate,'DD')	28

DDD	day num in year	to_char(sysdate,'DDD') 31+28+31+30+31+30+31+28 = 240	240
D	day num in week	to_char(sysdate,'D') 1 => sun 2 => mon . . 7 => sat	2
DY / dy	short weekday name	to_char(sysdate,'DY')	MON
DAY / day	full weekday name	to_char(sysdate,'DAY')	MONDAY
Q	quarter number jan to mar => 1 apr to jun => 2 jul to sep => 3 oct to dec => 4	to_char(sysdate,'Q')	3
CC	century number	to_char(sysdate,'CC')	21
AD / BC	AD or BC	to_char(sysdate,'BC')	AD
HH / HH12	hours in 12 hrs format		
HH24	hours in 24 hrs format		
MI	minutes part		
SS	seconds part		
FF	fractional seconds		
AM / PM	AM or PM		

Display current system time:

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

Display current system time in 24hrs format:

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

Display the emp records who joined in 1982:

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

(or)

```
SELECT ename,hiredate
FROM emp
WHERE hiredate BETWEEN '1-JAN-1982' AND '31-DEC_1982';
```

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

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

Display the emp records who joined in Jan and Dec:

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

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:

D	1
DY	SUN
DAY	SUNDAY

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

Output:

displays emp records joined on Sunday

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

Output:

displays emp records joined on Sunday

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

SUNDAY3spaces = SUNDAY FALSE

Output:

no rows selected

SUNDAY3spaces	**9**
MONDAY3spaces	**9**
TUESDAY2spaces	**9**
WEDNESDAY	**9**
THURSDAY1space	**9**
FRIDAY3spaces	**9**
SATURDAY1space	**9**

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

RTRIM('SUNDAY3spaces')
SUNDAY=SUNDAY TRUE

Output:
displays the emps joined on Sunday

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

SUNDAY3spaces = SUNDAY3spaces

Output:
displays the emps joined on Sunday

Display the emp records who joined in january:

```
SELECT ename,hiredate
FROM emp
WHERE to_Char(hiredate,'MONTH') = 'JANUARY';
```

Output:
no rows selected

JANUARY2spaces = JANUARY   FALSE

```
SELECT ename,hiredate
FROM emp
WHERE RTRIM(to_Char(hiredate,'MONTH')) = 'JANUARY';
```

to_Char()   [Number to Char]:

- **it can be used to convert number to string.**
- **It is used to apply currency formats.**

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**
9	**digit**
, / G	**thousand separator**
. / D	**decimal point**

display 5000 as $5000.00:

SELECT to_char(5000,'L9999.99') FROM dual;

display 5000 as USD5,000.00:

SELECT to_char(5000,'C9,999.99') FROM dual;

NLS PARAMETERS	DEFAULT VALUE
NLS_CURRENCY	**$**
NLS_ISO_CURRENCY	**AMERICA**

to see NLS PARAMETERS list login as DBA:
username: system
password: nareshit

SQL> show parameters
--displays all parameters

SQL> show parameters 'NLS'
--displays NLS parameters

display 5000 as ¥5000.00:

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

display 5000 as JPY5000.00:

SELECT to_char(5000,'C9999.99','NLS_ISO_CURRENCY=JAPAN') FROM dual;

display 5000 as RS5000.00:

```
SELECT to_char(5000,'L9999.99','NLS_CURRENCY=RS') FROM dual;
```

display 5000 as INR5000.00:

```
SELECT to_char(5000,'C9999.99','NLS_ISO_CURRENCY=INDIA') FROM dual;
```

Display all emp names and salaries. apply currency symbol $ and thousand separator for salary:

sal NUMBER(7,2)

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

to_date():

- **it can be used to convert string to date.**
- **to insert date value, we use it.**
- **to extract part of the date from specific date.**

Syntax:

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

Examples:

to_date('25-DEC-2022') **string**	**25-DEC-22** **date**
to_date('25 DECEMBER 2022')	**25-DEC-22**
to_date('DECEMBER 25 2022')	**ERROR**
to_date('DECEMBER 25 2022','MONTH DD YYYY')	**25-DEC-22**
to_Date('25/12/2022')	**ERROR**
to_Date('25/12/2022', 'DD/MM/YYYY')	**25-DEC-22**

Example:

```
CREATE TABLE demo
(
f1 DATE
);

INSERT INTO demo VALUES('25-DEC-2022');
```

string

DEMO

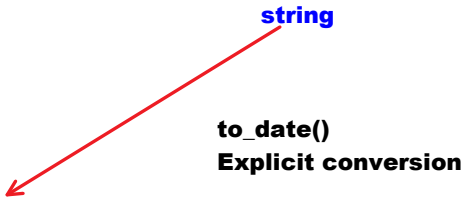
```
F1
--------
25-DEC-22
```

date

Implicit conversion

25-DEC-22 date

INSERT INTO demo VALUES(to_date('25-DEC-2022'));



F1

25-DEC-22 date

INSERT INTO demo VALUES(to_date('25/12/2022','DD/MM/YYYY'));

F1

25-DEC-22 date

INSERT INTO demo VALUES(to_date('&d/&m/&y','DD/MM/YYYY'));
Output:
enter value for d: 15
enter value for m:8
enter value for y:2023

/
Output:
enter value for d: ..
enter value for m:..
enter value for y:..

Display year part from today's date:

SELECT to_char(sysdate,'YYYY') FROM dual;
Output: date type
2023

Display year part from the date 18-OCT-2022:

SELECT to_char('18-OCT-2022','YYYY') FROM dual;
Output: string
ERROR

SELECT to_char(to_date('18-OCT-2022'),'YYYY') FROM dual;

Output:
2022

Find today's weekday name:

SELECT to_char(sysdate,'DAY') FROM dual;
Output:
TUESDAY

Find the weekday name on which india got independence:

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

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

TO_NUMBER():

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

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')	**ERROR**
to_number('USD5,000.00','C9,999.99')	**5000**

Aggregate Functions [Group Functions / Multi Row Functions]:

50
90
20
30

sum	**50+90+20+30 = 190**
avg	**190/4 = 47.5**
max	**90**
min	**20**
count	**4**

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 count no of records or no of column values

Syntax:
count(<column> / *)

Examples:

Find max salary in all emps:

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

Find max salary in all managers:

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

Find max salary in deptno 30:

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

Find min salary in all emps:

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

Find how much amount organization is spending on all emps [find sum of salaries of all emps]:

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

Find deptno 30 sum of salaries:

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

Find no of records in emp table:

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

Find how many emps are getting commission:

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

Date Functions:

sysdate
systimestamp

Add_Months()
Last_Day()
Next_Day()
Months_Between()

sysdate:

- **is used to get current system date**

systimestamp:

- **is used to get current system date and time**

Examples:
display current system date:

```
SELECT sysdate FROM dual;
```

display current system time using sysdate:

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

display current system date and time:

SELECT systimestamp FROM dual;

display to_day's date using systimestamp:

SELECT to_char(systimestamp,'DD/MM/YYYY') FROM dual;

Add_Months():

- used to add months to specific date.
- using it, we can also subtract months from specific date.

Syntax:

Add_Months(<date>, <no_of_months>)

Examples:
sysdate => 30-AUG-23

Add 2 days to sysdate:

SELECT sysdate+2 FROM dual;
Output:
1-SEP-23

Add 2 months to sysdate:

SELECT Add_Months(sysdate,2) FROM dual;
Output:
30-OCT-23

Add 2 years to sysdate:

SELECT Add_Months(sysdate,2*12) FROM dual;
Output:
30-AUG-25

Subtract 2 days from sysdate:

SELECT sysdate-2 FROM dual;
Output:
28-AUG-23

Subtract 2 months from sysdate:

SELECT Add_Months(sysdate,-2) FROM dual;
Output:
30-JUN-23

Subtract 2 years from sysdate:

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

Output:

30-AUG-21

Example:

ORDERS

ORDERID	ORDERED_DATE	DELIVERY_DATE	..	..
1234	sysdate	sysdate+5		

PRODUCTS

PID	MANUFACTURED_DATE	EXPIRY_DATE
1001	sysdate	Add_Months(manufactured_date,3)

CM_LIST

State_Code	CM_NAME	START_DATE	END_DATE
TS	KCR	17-AUG-2018	Add_Months(start_date,5*12)

EMPLOYEE

EMPID	ENAME	DOB	DOR
1001	A	15-AUG-2000	Add_Months(DOB,60*12)

DOB => dateOfBirth

DateOfRetirement

```
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));
```

Display the emp records who joined today:

```
SELECT ename,hiredate
FROM emp
WHERE hiredate=sysdate;
```

30-AUG-23 6:50 PM = 30-AUG-23 6:59 PM => FALSE

Output:

No rows selected

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

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

TRUNC(30-AUG-23 6:50 PM) = TRUNC(30-AUG-23 6:59 PM)
30-AUG-23 = 30-AUG-23 => TRUE

(or)

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

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 from sysdate:

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

Display the emp records who joined 1 year ago from sysdate:

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

Assignment:

SALES

DATEID	AMOUNT
1-JAN-2020	2500000
2-JAN-2020	2000000
..	
..	
30-AUG-2023	

Find today's sales:

```
select amount from salaes
where trunc(dateid) = trunc(sysdate);
```

Find yesterday sales

Find 2days ago sales

Find 1 month ago sales

Find 2 months ago sales

Find 1 year ago sales

Last_Day():
It is used to get last date in the month

Syntax:
Last_day(<date>)

Examples:
SELECT last_day(sysdate) FROM dual;
Output: 31-AUG-23

SELECT last_day('18-FEB-2023') FROM dual;
Output: 28-FEB-23

SELECT last_day('18-FEB-2020') FROM dual;
Output: 29-FEB-20

Find next month first date:

SELECT last_day(sysdate)+1 FROM dual;

31-AUG-23+1

Find current month 1st date:

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

Next_Day():
- **It is used to get coming date based on weekday**
- **For Example: coming Sunday date, coming Friday date**

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

Example:
Find coming Sunday date:

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

Months_Between():
used to get difference between 2 date values

Syntax:
Months_Between(<date1>, <date2>)

Example:
SELECT months_between('30-AUG-2023','30-AUG-2022')/12 FROM dual;
Output: 12

SELECT months_between('30-AUG-2023','30-AUG-2022')/12 FROM dual;

Output: 1

Find experience of all emps:

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

(or)

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

Number Functions:

sqrt()	ceil()
power()	floor()
sign()	trunc()
abs()	round()
	mod()

sqrt():
used to find square root value

Syntax:
sqrt(<number>)

Examples:

sqrt(100)	10
sqrt(25)	5

power():
used to find power value

Syntax:
power(<number>, <power>)

Example:

power(2,3)	8

sign():

- used to know sign of the number
- if number is +ve, returns 1
- if number is -ve, returns -1
- if number is 0, returns 0

Syntax:

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

abs():
- used to get absolute value
- absolute => non-negative

Syntax:

abs(<number>)

Examples:

abs(25)	25
abs(-25)	25

mod():
it used to get remainder value

Syntax:

mod(<number>, <divisor>)

Examples:

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

ceil():
used to get round up value

Syntax:

ceil(<number>)

floor():
used to get round down value

Syntax:

floor(<number>)

123 => 123.45678 => 124
floor() ceil()

floor(123.45678)	123
ceil(123.45678)	124

TRUNC():

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

Syntax:

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

Examples:

TRUNC(456.78923)	**456**
TRUNC(456.78923,1)	**456.7**
TRUNC(456.78923,2)	**456.78**
TRUNC(456.78923,3)	**456.789**

Note:

- **2nd argument can be given as negative.**
- **If 2nd argument is -ve, it does not give decimal places.**

When 2nd argument is -ve

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

TRUNC(678.45678,-1)	**670 and 680 670**
TRUNC(123.6542,-1)	**120 and 130 120**
TRUNC(456.789,-2)	**400 and 500 400**
TRUNC(2345.678,-3)	**2000 and 3000 2000**

ROUND():

- **if value is average or above average, it gives upper value**
- **if value is below average, it gives lower value**

Syntax:

ROUND(<number> , <no_of_decimal_places>)

Examples:

ROUND(123.4567)	**123 and 124 avrg => 123.5 123**
ROUND(123.7567)	**123 and 124 avrg => 123.5**

TRUNC(123.4567)	**123**
TRUNC(123.7567)	**123**

124

ROUND(123.45678,2)	123.46
ROUND(123.45378,2)	123.45

TRUNC(123.45678,2)	123.45
TRUNC(123.45378,2)	123.45

ROUND(123.456,-1)	120 and 130 avg => 125 120
ROUND(127.456,-1)	120 and 130 avg => 125 130
ROUND(567.8923,-2)	500 and 600 avrg => 550 600
ROUND(537.8923,-2)	500 and 600 avrg => 550 500

Analytic Functions:

RANK()
DENSE_RANK()
ROW_NUMBER()

RANK():

- **It is used to apply ranks to the records according to specific column order.**
- **If multiple values are same, it does not follow sequence in ranking [gaps will be there]**

Syntax:
RANK() OVER(PARTITION BY <column> ORDER BY <column> ASC / DESC)

DENSE_RANK():

- **It is used to apply ranks to the records according to specific column order.**
- **DENSE => No gaps**
- **Even if multiple values are same, it follows sequence in ranking. No gaps will be there in ranking.**

Syntax:
DENSE_RANK() OVER(PARTITION BY <column> ORDER BY <column> ASC / DESC)

Example:

MARKS	RANK	DENSE RANK
678	8	4
789	6	3
940	1	1

830	3	2
789	6	3
940	1	1
500	9	5
830	3	2
400	10	6
830	3	2

Apply ranks to emp records. give top rank to highest salary:

```
SELECT ename, sal,
RANK() OVER(ORDER BY sal DESC) as rnk
FROM emp;

(or)

SELECT ename, sal,
DENSE_RANK() OVER(ORDER BY sal DESC) as rnk
FROM emp;
```

RANK()	does not follow sequence in ranking when multiple values are same
DENSE_RANK()	follows sequence in ranking even if multiple values are same

Apply ranks to emp records according to seniority. Give top rank to most senior:

```
SELECT ename,hiredate,
DENSE_RANK() OVER(ORDER BY hiredate ASC) as rnk
FROM emp;
```

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

```
SELECT ename,sal,hiredate,
dense_rank() over(ORDER BY sal DESC, hiredate ASC) as rnk
FROM emp;
```

Note:
If salary same then only it checks seniority. to the senior gives top rank

PARTITION BY deptno

EMP

EMPNO	ENAME	SAL	DEPTNO	RANK
1001	A	5000	10	3
1002	B	7000	10	1

partition-1 **deptno 10**

partition-1	1001	A	5000	10	3	deptno 10
	1002	B	7000	10	1	
	1003	C	6000	10	2	
partition-2	1004	D	10000	20	1	deptno 20
	1005	E	5000	20	3	
	1006	F	8000	20	2	

PARTITION BY clause:

- **PARTITION BY clause is used to group the records according to specific column.**
- **Within the groups ranks will be applied**

Example:
Apply ranks to emp records within dept. give top rank highest salary:

```
break on deptno skip 1              --deptno 10 will not be duplicated

break on deptno skip 1 duplicates   --deptno 10 will be duplicated
```

```
SELECT ename,sal,
dense_rank() over(PARTITION BY deptno ORDER BY sal DESC) as rnk
FROM emp;
```

ROW_NUMBER():
Used to apply row numbers to the records

Syntax:

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

Example:
Display row numbers to all records. According to empno Ascending Order apply the row numbers:

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

Apply row numbers to the records according to sal descending order:

```
SELECT row_number() over(order by sal desc) as sno,
empno,ename,sal
FROM emp;
```

Apply row numbers to the records with in dept:

```
SELECT row_number() over(PARTITION BY deptno ORDER BY sal DESC) as sno,
empno,ename,deptno
FROM emp;
```

Miscellaneous Functions:

NVL()
NVL2()
COALESCE()

USER
UID

GREATEST()
LEAST()

NVL():

- **It is replace the nulls.**

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 on NVL():

Calculate total salary of all emps [total salary = sal+comm]:

```
SELECT ename,sal,comm,
sal+NVL(comm,0) as "total salary"
FROM emp;
```

Display emp names, salaries and commission. If commission is null display it as 'N/A' [Not Applicable]:

```
SELECT ename,sal,NVL(comm,'N/A') as comm
FROM emp;
```

Output:
ERROR

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

Output:

replaces nulls with N/A in comm column

NVL2():

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

Syntax:

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

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

Examples:

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

Update comm of all emps as following:
if employee is getting commision, increase 1000 rupees comm
if employee is not getting comm, set comm as 700

```
UPDATE emp
SET comm=NVL2(comm,comm+1000,700);
```

NVL()	•can replace nulls only •can take 2 arguments
NVL2()	can replace nulls and not nulls •can take 3 arguments

Coalesce():
returns first not null value

Syntax:

Colesce(v1, v2, v3, ..., v_n)

Examples:

Coalesce(null,null,null,500,null,600,700)	500

Example on Coalesce():

Customer

CID	CNAME	MOBILE1	MOBILE2
1001	A		9123456789

1002	B	8912345678	
1003	C	7123456789	6123456789
1004	D		7891235436

```
SELECT cid,cname,coalesce(mobile1,mobile2) as mobile
FROM Customer;
```

1001	A	9123456789
1002	B	8912345678
1003	C	7123456789
1004	D	7891235436

USER:
it returns current user name

Example:

display current user name:

```
SELECT user FROM dual;
(or)
SHOW USER
```

UID:
it returns user id

Example:

Display current user id and name:

```
SELECT uid, user FROM dual;
```

to see all users list:

```
DESC all_users;

SELECT user_id, username FROM all_users;
```

all_users	is a system / built-in table. it maintains all users information

MAX():
is used to find max value in vertical values

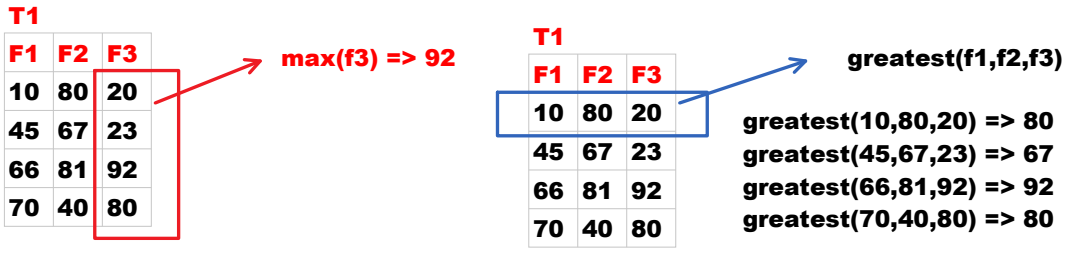
GREATEST():
is used to find max value in horizontal values

Syntax:

```
GREATEST(v1, v2, v3, ..., v_n)
```

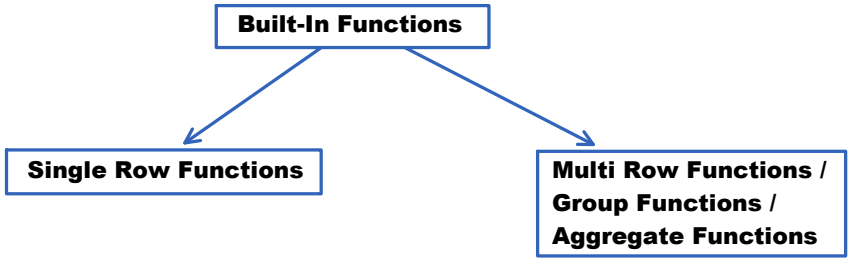
Examples:

GREATEST(10,20,30)	30
GREATEST(50,90,20,30,70,60)	90



max()	it is used to find max value in column multi row function. 1 function call applied on multiple rows can take 1 argument
greatest()	it is used to find max value in row single row function. 1 function call will be applied on 1 row any no of arguments [variable length argument]

min()	used to find min value in column multi row function. can take 1 argument
least()	used find min value in row single row function. any no of arguments [variable length argument]



Examples:
String Functions
Conversion Functions
Date Functions

Examples:
max()
min()
avg()

String Functions
Conversion Functions
Date Functions
Analytic Functions
Number Functions

Examples:
max()
min()
avg()
sum()
count()

Single row function
means, 1 function call
will be applied on 1 row

Multi row function
means, 1 function call
will be applied on multiple rows

ename	lower(ename)
SMITH	lower('SMITH')
ALLEN	lower('ALLEN')
WARD	lower('WARD')

sal	max(sal) => 6000
4000	
6000	
3000	

CLAUSES in SELECT command

Saturday, September 2, 2023 6:19 PM

CLAUSES in SELECT command:

Syntax of SELECT command:

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

SQL	ENGLISH
QUERIES	SENTENCES
CLAUSES	WORDS

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

SELECT command clauses are:

- SELECT
- FROM
- WHERE
- ORDER BY
- DISTINCT
- GROUP BY
- HAVING
- OFFSET
- FETCH

Display the emp records whose salary is more than 2500:

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

SELECT clause	is used to specify column list Example: SELECT ename,sal
FROM clause	is used to specify table list Examples:

	FROM emp FROM emp,dept
WHERE clause	• is used specify filter condition • WHERE clause condition will be applied on every row • It filters the rows Examples: WHERE job='MANAGER' WHERE sal>2500

ORDER BY:

- **ORDER BY clause is used to arrange the records in ascending or descending order according to specific column / columns.**
- **Default Order is: ASC**

Syntax:

```
ORDER BY <column> ASC/DESC [, <column> ASC/DESC , .....]
```

Number		CHAR		DATE ASC
1,2,3,4, ... 10	ASC	A to Z	ASC	1-JAN-2020
10,9,8,..... 1	DESC	Z to A	DESC	2-JAN-2020
				.
				.
				31-DEC-2020
				1-JAN-2021
				.

Examples on ORDER BY:

Display all emp records. arrange them in alphabetical order according emp name:

```
ename
----------
WARD
ALLEN
SMITH
BLAKE
ADAMS
```

```
ADAMS
ALLEN
BLAKE
SMITH
WARD
```

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

(or)

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

(or)

```
SELECT ename,sal
FROM emp
ORDER BY 1 ASC;
```

1	ename
2	sal

here 1,2 are column numbers
in SELECT clause

Display all emp records. display highest salary first.
[display records in descending order according to salary]:

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

(or)

```
SELECT ename,sal
FROM emp
ORDER BY 2 DESC;
```

Display all emp records. Display most senior record first:

```
hiredate                  hiredate ASC
---------------
25-DEC-1982               15-AUG-1980
15-AUG-1980               25-DEC-1982
```

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

Display all emp records. Arrange them in Ascending
Order according to deptno:

```
break on deptno skip 1 duplicates

SELECT empno,ename,deptno,sal
```

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

Display the emp records in ascending order according to deptno. Within the dept arrange salaries in descending order:

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

case-1:	20	10000	10	6000
	10	6000	20	10000
case-2:				
	10	3000	10	6000
	10	6000	10	3000

If deptno is different [case-1], it will not check salary
If deptno is same [case-2], it checks salary

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

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

sal
5000
8000
6000
9000
7000
3000
4000

ORDER BY sal DESC

9000
8000
7000
6000
5000
4000
3000

in ASCENDING	nulls will be given last
in DESCENDING	nulls will be given first

4000

5000
4000
3000

Display all emp records. arrange them according to salary descending order. display nulls last:

```
SELECT ename,sal
FROM emp
ORDER BY sal DESC nulls last;
```

Display all emp records. arrange them in ascending order according to salary. display nulls first:

```
SELECT ename,sal
FROM emp
ORDER BY sal ASC nulls first;
```

DISTINCT clause:

- **it is used to avoid [eliminate] duplicate rows.**

Syntax:

```
SELECT DISTINCT <column>
```

Examples on DISTINCT:

Display the job titles offered by company:

```
SELECT job FROM emp;
(or)
SELECT ALL job FROM emp;
```

```
job
-----
MANAGER
CLERK
MANAGER
CLERK
CLERK
SALESMAN
SALESMAN
ANALYST
MANAGER
ANALYST
SALESMAN
```

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

```
job
---------
MANAGER
CLERK
SALESMAN
ANALYST
```

Display the deptnos available in emp table:

```
SELECT deptno FROM emp;
(or)
SELECT ALL deptno FROM emp;
```

```
deptno
----------
20
30
30
20
20
30
10
10
20
10
30
```

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

```
10
20
30
```

GROUP BY:

- It is used to group the records according to specific column / columns.
- We can apply aggregate [group] functions on groups.
- It is mainly used for data analysis.
- It gives summarized data from detailed data.

Example:

GROUP BY deptno

Detailed Data → **Summarized data**

EMP

EMPNO	ENAME	DEPTNO	SAL
1001	A	10	10000
1002	B	20	15000
1003	C	20	5000
1004	D	10	20000
1005	E	30	5000
1006	F	30	10000

deptno	sum(Sal)
10	30000
20	20000
30	15000

Examples on GROUP BY:

Find dept wise sum of salaries:

DEPTNO	SUM_OF_SAL
10	?
20	?
30	?

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

Display 10th dept & 30th dept sum of salaries:

deptno	sum_of_Sal
10	?
30	?

```
SELECT deptno, sum(sal) AS sum_of_sal
FROM emp
WHERE deptno IN(10,30)
GROUP BY deptno
ORDER BY deptno;
```

Execution order of clauses:

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

FROM emp:
selects entire emp table

EMP

EMPNO	ENAME	DEPTNO	SAL
1001	A	30	10000
1002	B	20	15000
1003	C	20	5000
1004	D	30	20000
1005	E	10	5000
1006	F	10	10000

WHERE deptno IN(10,30):
It filters the rows

1001	A	30	10000
1004	D	30	20000
1005	E	10	5000
1006	F	10	10000

GROUP BY deptno:
it groups the records according to deptno
On result of GROUP BY aggregate function will be applied

1001	A	30	10000
1004	D	30	20000

30 group => sum(sal) => 30000

1005	E	10	5000
1006	F	10	10000

10 group => sum(sal) => 15000

SELECT deptno, sum(sal) AS sum_of_sal:

It selects the columns

30	30000
10	15000

ORDER BY deptno:
it arranges in ascending order according to deptno

10	15000
30	30000

Find Dept wise number of emps:

DEPTNO	NO_OF_EMPS
10	?
20	?
30	?

```
SELECT deptno, count(*) as no_of_emps
FROM emp
GROUP BY deptno
ORDER BY deptno;
```

Find year wise no of emps joined in organization:

YEAR	no_of_emps
1980	?
1981	?
1982	?
1983	?

```
SELECT to_Char(hiredate,'YYYY') AS year, count(*) AS no_of_emps
FROM emp
GROUP BY to_Char(hiredate,'YYYY')
ORDER BY 1;
```

Find quarter wise no of emps joined in organization:

Quarter	no_of_emps
1	?
2	?
3	?
4	?

```
COLUMN quarter FORMAT A7

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

Assignment:
Find month wise no of emps joined in organization:

MONTH	NO_OF_EMPS
JAN	?
FEB	?
MAR	?
..	
DEC	?

Find weekday wise no of emps joined in organization:

Weekday	No_of_emps
sun	?
mon	?
..	
sat	?

Find max salary and min salary in each dept:

deptno	max_Sal	min_sal
10	?	?
20	?	?
30	?	?

```
SELECT deptno, max(Sal) as max_Sal,
min(sal) as min_sal
FROM emp
GROUP BY deptno
ORDER BY 1;
```

Find jobwise sum of salaries:

JOB	sum_of_sal
CLERK	?
MANAGER	?
ANALYST	?

```
GROUP BY job
sum(sal)
```

```
SELECT job, sum(sal)
FROM emp
GROUP BY job;
```

Assignment

Find job wise max salary and min salary:

JOB	MAX_SAL	MIN_SAL
CLERK	?	?
MANAGER	?	?

```
GROUP BY job
max(sal)
min(sal)
```

Find jobwise no of emps

```
GROUP BY job
```

JOB	NO_OF_EMPS
CLERK	?
MANAGER	?

GROUP BY job
count(*)

```
SELECT to_char(hiredate,'YYYY') as year,
count(*) as no_of_emps
FROM emp
GROUP BY to_char(hiredate,'YYYY')
ORDER BY year;
```

Output:
year wise no of emps

```
SELECT to_char(hiredate,'YYYY') as year,
count(*) as no_of_emps
FROM emp
GROUP BY year
ORDER BY year;
```

Output:
ERROR: YEAR invalid identifier [GROUP BY year]

Can We use column alias in ORDER BY?
Yes. ORDER BY gets executed after SELECT

Can we use column alias in GROUP BY?
No. GROUP BY gets executed before SELECT [oracle 21c]

GROUPING RECORDS ACCORDING TO MULTIPLE COLUMNS:

GROUP BY deptno,job

empno	ename	deptno	job
1001	A	20	ANALYST
1002	B	20	ANALYST
1003	C	30	SALESMAN
1008	D	30	SALESMAN
1004	E	10	CLERK
1005	F	10	CLERK
1006	G	20	CLERK
1007	H	20	CLERK

20 ANALYST

30 SALESMAN

10 CLERK

20 CLERK

1007	H	20	CLERK

Find dept wise, job wise sum of salaries:

DEPTNO	JOB	SUM_OF_SAL
10	CLERK	?
10	MANAGER	?
20	ANALYST	?
20	CLERK	?
20	MANAGER	?
30	SALESMAN	?
30	CLERK	?
30	MANAGER	?

```
SELECT deptno,job,sum(sal)
FROM emp
GROUP BY deptno,job
ORDER BY 1;
```

Rollup() & Cube():

Rollup() & Cube() functions are used to calculate sub totals and grand total.

Rollup() syntax:

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

Example:

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

It calculates sub totals according to deptno only

Cube() syntax:

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

Example:

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

It calculates dept wise sub totals and job wise sub totals

ROLLUP()	it calculates subtotal according to first column in grouping column list
CUBE()	it calculates subtotal according to all columns in grouping column list

Examples:

Find dept wise, job wise sum of salaries.

Calculate sub totals and grand total according to deptno:

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

DEPTNO	JOB	SUM(SAL)
10	CLERK	10000
10	MANAGER	20000
	10 sub total	30000
20	CLERK	5000
20	MANAGER	15000
	20 sub total	20000
	GRAND TOTAL	50000

Find dept wise, job wise sum of salaries.
Calculate dept wise subtotals and job wise subtotals and grand total:

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

DEPTNO	JOB	SUM(SAL)
10	CLERK	10000
10	MANAGER	20000
	10 sub total	30000
20	CLERK	5000
20	MANAGER	15000
	20 sub total	20000
	CLERK SUBTOTAL	15000
	MANAGER SUBTOTAL	35000
	GRAND TOTAL	50000

Display year wise, quarter wise no of emps joined in organization:

year	quarter	no_of_emps
1980	1	?
	2	?
	3	?
	4	?
1981	1	?
	2	?
	3	?
	4	?

```
break on year skip 1


SELECT to_char(hiredate,'YYYY') as year,
to_char(hiredate,'Q') as quarter,
count(*) as no_of_emps
FROM emp
GROUP BY to_char(hiredate,'YYYY'), to_char(hiredate,'Q')
ORDER BY year;
```

Display year wise, quarter wise no of emps joined in organization.
calculate year wise subtotals and grand total:

1980	1	2
	2	3
	3	1
	4	2
	1980 subtotal	8
1981	1	..
	2	..
	3	..
	4	..
	1981 subtotal	
	grand total	

```
SELECT to_char(hiredate,'YYYY') as year,
to_char(hiredate,'Q') as quarter,
count(*) as no_of_emps
FROM emp
GROUP BY Rollup(to_char(hiredate,'YYYY'), to_char(hiredate,'Q'))
ORDER BY year;
```

Display year wise, quarter wise no of emps joined in organization.
calculate year wise subtotals, quarter wise subtotals and grand total:

1980	1	2
	2	3
	3	1
	4	2
	1980 subtotal	8
1981	1	..
	2	..
	3	..
	4	..
	1981 subtotal	--
	1st quarter subtoal	..
	2nd subtotal	..
	3rd subtotal	
	4th subtotal	
	grand total	

```
SELECT to_char(hiredate,'YYYY') as year,
to_char(hiredate,'Q') as quarter,
count(*) as no_of_emps
FROM emp
GROUP BY Cube(to_char(hiredate,'YYYY'), to_char(hiredate,'Q'))
ORDER BY year;
```

Assignment: **calculate year wise, quarter wise sales**

SALES

DATEID	AMOUNT
1-JAN-2020	200000
2-JAN-2020	180000

2020	1	?
	2	?
	3	?
	4	?

```
sum(amount)
GROUP BY year, quarter
```

2-JAN-2020	180000
..	
..	
5-SEP-2023	2500000

	4	?
2021	1	?
	2	?
	3	?
	4	?

calculate year wise, quarter wise sales
calculate subtotals according to year
[Rollup()]

calculate year wise, quarter wise sales
calculate year wise subtotals, quarter wise subtotals and grand total
[Cube()]

Assignment:

PERSON

PID	PNAME	STATE	GENDER	AADHAR
1001	A	TS	M	..
1002	B	TS	M	..
1003	C	TS	F	
1004	D	TS	F	
1005	E	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, gender wise population
[GROUP BY state, gender]

TS	M	..
TS	F	..
AP	M	..
AP	F	..

Find state wise, gender wise population.
calculate subtotal according to state
[GROUP BY ROLLUP(state, gender)]

STATE	GENDER	no_of_people
TS	M	?
	F	?

	TS subtotal	?
AP	M	?
	F	?
	AP sub total	?
	Grand total [india population	

Find state wise, gender wise population.
calculate subtotal according to state and gender
[GROUP BY CUBE(state, gender)]

STATE	GENDER	no_of_people
TS	M	?
	F	?
	TS subtotal	?
AP	M	?
	F	?
	AP sub total	?
	M subtotal	?
	F subtotal	?
	GRAND TOTAL	?

HAVING:

- HAVING clause is used to write conditions on groups.
- It filters the groups.
- It will be applied on result of GROUP BY.
- It cannot be used without GROUP BY.
- Aggregate Functions [Group Functions] can be used in HAVING clause.

Example:

Display the depts which are spending more than 10000:

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

Display the deptnos which are having 5 or more emps:

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

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

```
SELECT job, count(*)
FROM emp
GROUP BY job
HAVING count(*)<3;
```

Differences between WHERE and HAVING:

WHERE	HAVING
WHERE clause condition will be applied on every row	HAVING clause condition will be applied on every group
It filters the rows	It filters the groups
We cannot use aggregate functions in WHERE clause	We can use aggregate functions in HAVING clause
It can be used without GROUP BY	It cannot be used without GROUP BY
WHERE gets executed before GROUP BY	HAVING gets executed after GROUP BY

Execution order of SELECT command clauses:

FROM
WHERE
GROUP BY
HAVING
SELECT
DISTINCT
ORDER BY
OFFSET
FETCH

Note:

```
SELECT deptno, ename, sum(Sal)
FROM emp
GROUP BY deptno;

Output:
ERROR: ename is not a GROUP BY expression
```

When GROUP BY is used, we can specify GROUP BY columns or Group Functions in SELECT clause. Other than these cannot specified in SELECT clause.

OFFSET clause:

- introduced in ORACLE 12C version.
- it is used to specify no of rows to be skipped

Syntax:

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

FETCH clause:

- introduced in ORACLE 12C version.
- it is used to specify no of rows to be fetched [selected]

Syntax:

```
FETCH FIRST/NEXT <number>
```

EMP

EMPNO	ENAME	SAL	DEPTNO
7369	SMITH	800	20
7499	ALLEN	1600	30
7521	WARD	1250	30
7566	JONES	2975	20
7782	CLARK	2450	10
7934	MILLER	1300	10
1001	A	1800	
1002	B	2000	

DEPT

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

JOINS

Wednesday, September 6, 2023 6:53 PM

JOINS:

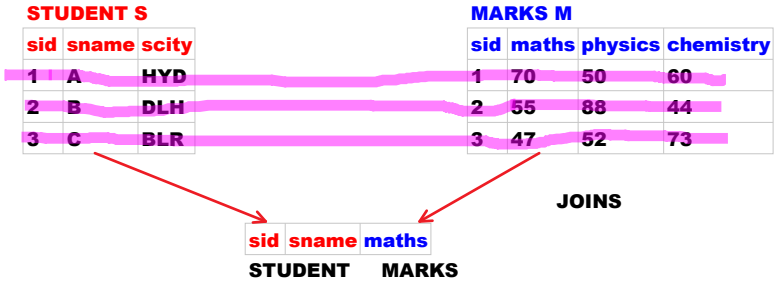
GOAL:
it is used to retrieve the data from multiple tables

JOIN => connect/combine/link

COLLEGE DB

STUDENT
MARKS
FEE
STAFF
.
.

S.sid = M.sid => Join Condition



JOINS:

- **JOIN => connect / combine / link**
- **JOIN is an operation like filtering, sorting.**
- **In Join Operation, one table record will be joined [combined] with another table record based on some condition. This condition is called "Join Condition". This operation is called "Join Operation".**
- **JOINS concept is used to retrieve the data from multiple tables.**
- **JOIN condition decides which record in one table should be joined with which record in another table.**

Types Of Joins:

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

Equi Join:

- **If Join operation is performed based on equality**

condition then it is called "Equi Join".

Example on Equi Join:

S.sid = M.sid JOIN CONDTION

STUDENT S

sid	sname	scity
1	A	HYD
2	B	DLH
3	C	BLR

MARKS M

sid	maths	physics	chemistry
1	70	50	60
2	55	88	44
3	47	52	73

Display student details with maths subject marks:

sid	sname	maths

STUDENT MARKS

```
CREATE TABLE student
(
sid NUMBER(4),
sname VARCHAR2(10),
scity CHAR(3)
);

CREATE TABLE marks
(
sid NUMBER(4),
maths NUMBER(3),
physics NUMBER(3),
chemistry NUMBER(3)
);

INSERT INTO student VALUES(1,'A','HYD');
INSERT INTO student VALUES(2,'B','DLH');
```

1	70	50	60
2	55	88	44

```
INSERT INTO marks VALUES(1,70,50,60);
INSERT INTO marks VALUES(2,55,88,44);

COMMIT;

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

e.deptno = d.deptno Join Condition

EMP e

EMPNO	ENAME	SAL	DEPTNO
7369	SMITH	800	20
7499	ALLEN	1600	30
7521	WARD	1250	30
7566	JONES	2975	20
7782	CLARK	2450	10
7934	MILLER	1300	10
1001	A	1800	
1002	B	2000	

DEPT d

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

unmatched record from DEPT

7934	MILLER	1300	10
1001	A	1800	
1002	B	2000	

unmatched record from DEPT

unmatched records from EMP

Display emp details along with dept details [Equi Join]:

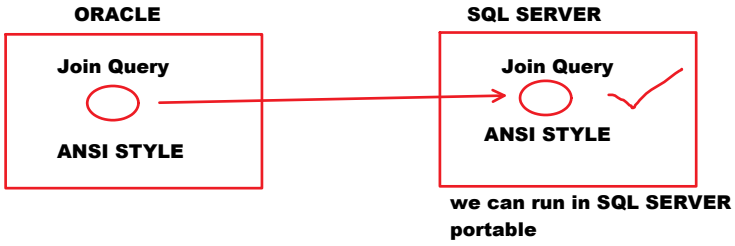
ename	sal	dname	loc

EMP e DEPT d

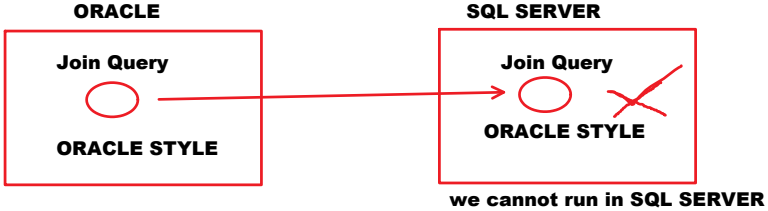
```
SELECT e.ename,e.sal.d.dname,d.loc
FROM emp e, dept d
WHERE e.deptno = d.deptno;
```

From ORACLE 9i version onwards, we can write a Join Query in 2 styles. They are:

- **ORACLE STYLE / NATIVE STYLE**
- **ANSI STYLE [best way => portable]**



we can run in SQL SERVER
portable



we cannot run in SQL SERVER
not portable

Note:
In ORACLE STYLE, to separate two table names, use , [comma]
In ANSI STYLE, to separate two table names, use keyword

In ORACLE STYLE, write join condition in WHERE clause
In ANSI STYLE, write join condition in ON clause

Display emp details along with dept details [Equi Join]:

ename	sal	dname	loc

EMP e DEPT d

ORACLE STYLE:

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

ANSI STYLE:

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

SET AUTOTRACE ON EXPLAIN --displays execution plan

Display BLAKE record along with dept details:

ORACLE STYLE:

ename	dname
EMP e	DEPT d

```
SELECT e.ename, d.dname
FROM emp e, dept d
WHERE e.deptno=d.deptno AND e.ename='BLAKE';
```

ANSI STYLE:

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

ON clause is used to specify Join Condition
WHERE clause is used to specify filter condition

e.deptno = d.deptno

EMP e

EMPNO	ENAME	SAL	DEPTNO
7369	SMITH	800	20
7499	ALLEN	1600	30
7521	WARD	1250	30
7698	BLAKE	2975	30
7782	CLARK	2450	10
7934	MILLER	1300	10
1001	A	1800	
1002	B	2000	

DEPT d

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

Display the emp records who are working in New York:

ename	sal	dname	loc
emp e		dept d	

ORACLE STYLE:

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

ANSI STYLE:

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

WHERE d.loc='NEW YORK';

e.deptno=d.deptno

EMP e

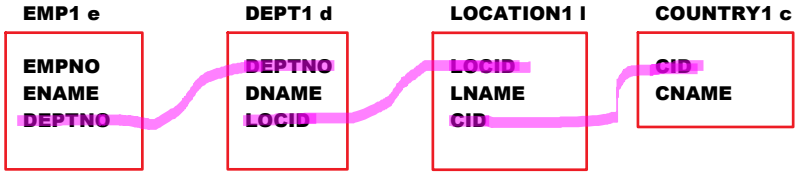
EMPNO	ENAME	SAL	DEPTNO
7369	SMITH	800	20
7499	ALLEN	1600	30
7521	WARD	1250	30
7698	BLAKE	2975	30
7782	CLARK	2450	10
7934	MILLER	1300	10
1001	A	1800	
1002	B	2000	

DEPT d

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

Retrieving data from 4 tables:

e.deptno = d.deptno d.locid=l.locid l.cid=c.cid



ename	dname	lname	cname
EMP1	DEPT1	LOCATION1	COUNTRY1

ORACLE STYLE:

```
SELECT e.ename, d.dname, l.lname, c.cname
FROM emp1 e, dept1 d, location1 l, country1 c
WHERE e.deptno=d.deptno AND d.locid=l.locid AND l.cid=c.cid;
```

ANSI STYLE:

```
SELECT e.ename,d.dname,l.lname,c.cname
FROM emp1 e INNER JOIN dept1 d
ON e.deptno=d.deptno INNER JOIN location1 l
ON d.locid=l.locid INNER JOIN country1 c
ON l.cid=c.cid;
```

Outer Join:

- Inner Join can give matched records only.
- Inner Join cannot give unmatched records.
- To get unmatched records also, we use Outer Join

- Outer Join can give matched records and unmatched records

Inner Join = matched records only

Outer Join = matched + unmatched records

There are 3 sub types in Outer Join. They are:

- Left Outer Join
- Right Outer Join
- Full Outer Join

Left Outer Join:

Left Outer join = matched + unmatched from left table

- **Left Outer Join can give matched records and unmatched records from left table**
- **For outer Join, we use outer join operator in oracle style.**
- **Outer join operator symbol is: (+)**
- **in ORACLE STYLE,**
  For Left outer Join write (+) symbol at right side.
- **in ANSI STYLE,**
  use the keyword: LEFT [OUTER] JOIN

Note:
In ORACLE STYLE, join condition decides left table and right table.

Example:

EMP	**e**
DEPT	**d**

WHERE e.deptno = d.deptno

e	**Left Table**
d	**Right Table**

WHERE d.deptno = e.deptno

d	**Left table**
e	**Right table**

In ANSI STYLE, keyword decides left table and right table

Example:
FROM emp e INNER JOIN dept d

emp	**left table**
dept	**right table**

FROM dept d INNER JOIN emp e

dept	**Left table**
emp	**Right table**

Display the emp records along with dept details.
Also display the emp records to whom dept is not assigned:
[Left Outer Join]

ename	**sal**	**dname**	**loc**

emp e dept d

```
INSERT INTO emp(empno,ename,sal) VALUES(1001,'A',5000);
INSERT INTO emp(empno,ename,sal) VALUES(1002,'B',6000);
COMMIT;
```

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 = matched + unmatched from right table**

- **Right Outer Join can give matched records and unmatched records from right table.**
- **in ORACLE STYLE,**
  for right outer join write (+) symbol at left side.
- **in ANSI STYLE,**
  use the keyword: RIGHT [OUTER] JOIN

Example:

Display the emp records along with dept details.
Also display the depts in which emps are not existed:
[Right Outer join]:

ename	sal	dname	loc

emp e dept d

ORACLE STYLE:

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

ANSI STYLE:

```
SELECT e.ename,e.sal,d.dname,d.loc
FROM emp e RIGHT OUTER JOIN dept d
ON e.deptno = d.deptno;
```

Full Outer Join:

- **Full Outer = matched + unmatched from left + unmatched from rigtht**

- **Full Outer Join can give matched records, unmatched records from left table and right table.**

A = {1,2,3,4,5}
B = {4,5,6,7,8}

- **Full Outer Join can give matched records, unmatched records from left table and right table.**

- **In ORACLE STYLE,**

Left Outer Join	**WHERE e.deptno=d.deptno(+)**
Right Outer Join	**WHERE e.deptno(+)=d.deptno**
Full Outer Join	**WHERE e.deptno(+) = d.deptno(+) => ERROR**
Full Outer Join	**LEFT OUTER JOIN UNION RIGHT OUTER JOIN**

A = {1,2,3,4,5}
B = {4,5,6,7,8}

A U B = {1,2,3,4,5,6,7,8}

Left Outer = matched + unmatched from left
Right Outer = matched + unmatched from right

Left Outer Join
Union
Right outer Join

matched + unmatched from left + unmatched from right

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

Example:

Display the emp records along with dept details.
Also display the emps to whom dept is not assigned.
Also display the depts which are not having emps:
[FULL OUTER JOIN]

ORACLE STYLE:

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

ANSI STYLE:

```
SELECT e.ename,e.sal,d.dname,d.loc
FROM emp e FULL OUTER JOIN dept d
ON e.deptno=d.deptno;
```

Displaying unmatched records only from the outer join:

Display the emp records to whom dept is not assigned:
[unmatched from emp]
[Left Outer Join + Condition] => gives unmatched only from left table

ORACLE STYLE:

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

ANSI STYLE:

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

Display the depts in which emps are not existed:
[Right Outer Join + Condition] => gives unmatched only from right table

ORACLE STYLE:

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

ANSI STYLE:

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

Display the emps to whom dept is not assigned.
Also display the depts in which emps are not existed:
[FULL OUTER + conditions] => gives unmatched only from left and right

ORACLE STYLE:

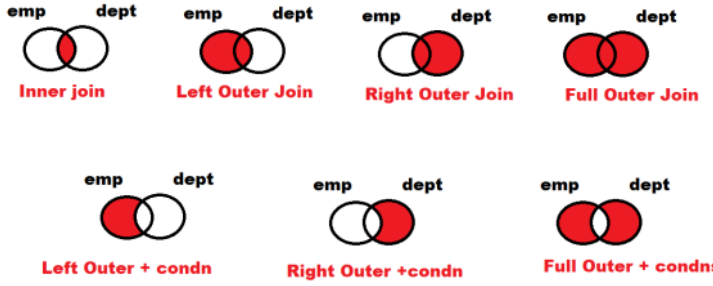
```
LEFT OUTER JOIN + condition
UNION
RIGHT OUTER JOIN + condition
```

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

ANSI STYLE:

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

Venn Diagrams of Joins:



cartesian product

A = {1,2,3}
B = {4,5}

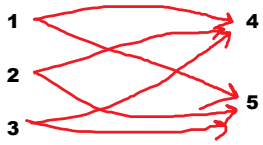
AXB =

Cross Join [Cartesian Join]:

- **In cross join, each record in first table will be joined**

with every record in second table.

- Don't write Join Condition for Cross Join.



{ (1,4)(1,5)
(2,4)(2,5)
(3,4)(3,5) }

Example:

GROUPA A

cid	cname
10	IND
20	AUS
30	WIN

GROUPB B

cid	cname
40	NZ
50	ENG
60	BAN

IND VS NZ
IND VS ENG
IND VS BAN

AUS VS NZ
AUS VS ENG
AUS VS BAN

WIN VS NZ
WIN VS ENG
WIN VS BAN

ORACLE STYE:

```
SELECT A.cname || ' VS ' || B.cname
FROM groupa A, groupb B;
```

ANSI STYLE:

```
SELECT A.cname || ' VS ' || B.cname
FROM groupa A CROSS JOIN groupb B;
```

Equi Join:
If Join operation is performed based on equality condition
then it is called "Equi Join".

Example:
WHERE e.deptno = d.deptno

Non-Equi Join / Inner Join:
If Join operation is performed based on other than equality condition
then it is called "Non-Equi Join".

Example:
WHERE e.deptno > d.deptno
WHERE e.deptno < d.deptno
WHERE e.deptno != d.deptno

Example:

e.sal BETWEEN s.losal AND s.hisal

EMP e

EMPNO	ENAME	SAL
1001	A	2500
1002	B	1300
1003	C	5000
1004	D	1000
1005	E	1800
1006	F	1500
1007	G	

SALGRADE s

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

Display the emp details along with salary grade:

ename	sal	grade

emp e salgrade s

ORACLE STYLE:

```
SELECT e.ename,e.sal,s.grade
FROM emp e, salgrade s
WHERE e.sal BETWEEN s.losal AND s.hisal;
```

ANSI STYLE:

```
SELECT e.ename,e.sal,s.grade
FROM emp e INNER JOIN salgrade s
ON e.sal BETWEEN s.losal AND s.hisal;
```

Self Join / Recursive Join:

EMP

EMPNO	ENAME	JOB	SAL	MGR
1001	A	MANAGER	15000	
1002	B	CLERK	5000	1001
1003	C	ANALYST	3000	1001
1004	D	MANAGER	20000	
1005	E	CLERK	6000	1004
1006	F	SALESMAN	4000	1004

emp_name	emp_sal	mgr_name	mgr_sal
B	5000	A	15000
C	3000	A	15000
E	6000	D	20000
F	4000	D	20000

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

Example on Self Join:

Display emp details along with manager details:
[Self Join]

emp_name	emp_sal	mgr_name	mgr_sal

e.MGR = m.EMPNO

EMP e

EMPNO	ENAME	JOB	SAL	MGR
1001	A	MANAGER	15000	
1002	B	CLERK	5000	1001
1003	C	ANALYST	3000	1001
1004	D	MANAGER	20000	
1005	E	CLERK	6000	1004

EMP m

EMPNO	ENAME	JOB	SAL	MGR
1001	A	MANAGER	15000	
1002	B	CLERK	5000	1001
1003	C	ANALYST	3000	1001
1004	D	MANAGER	20000	
1005	E	CLERK	6000	1004

1003	C	ANALYST	3000	1001
1004	D	MANAGER	20000	
1005	E	CLERK	6000	1004
1006	F	SALESMAN	4000	1004

1003	C	ANALYST	3000	1001
1004	D	MANAGER	20000	
1005	E	CLERK	6000	1004
1006	F	SALESMAN	4000	1004

ORACLE STYLE:

```
SELECT e.ename AS emp_name,
e.sal AS emp_sal,
m.ename AS mgr_name,
m.sal AS mgr_sal
FROM emp e, emp m
WHERE e.mgr = m.empno;
```

ANSI STYLE:

```
SELECT e.ename AS emp_name,
e.sal AS emp_sal,
m.ename AS mgr_name,
m.sal AS mgr_sal
FROM emp e INNER JOIN emp m
ON e.mgr = m.empno;
```

Display the emp records who are earning more than their manager:

emp_name	emp_sal	mgr_name	mgr_sal

ORACLE STYLE:

```
SELECT e.ename AS emp_name,
e.sal AS emp_sal,
m.ename AS mgr_name,
m.sal AS mgr_sal
FROM emp e, emp m
WHERE e.mgr = m.empno AND e.sal>m.sal;
```

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 emp records who are working under BLAKE:

emp_name	mgr_name
e.ename	m.ename

ORACLE STYLE:

```
SELECT e.ename AS emp_ename,
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_ename,
m.ename AS mgr_name
FROM emp e INNER JOIN emp m
ON e.mgr=m.empno
WHERE m.ename='BLAKE';
```

x.cid < y.cid

Example:

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
ONx.cid<y.cid;
```

JOINS
Goal: used to retrieve data from multiple tables

Types Of Joins:

Inner Join	**matched records only**	
	Equi Join	**based on equality condn join operation will be performed**
	Non-Equi Join	**based on other than equality condition join operation will be performed**
Outer Join	**matched + unmatched**	
	Left Outer	**matched + unmatched from left**
	Right Outer	**matched + unmatched from right**
	Full Outer	**matched + unmatched from L& R**
Self-Join		**table will be joined to itself one record in a table will be joined with another record in same table**
Cross Join		**each record in one table will be joined with every record in another table don't write join condition**

Assignment:

EMPLOYEE **PROJECT**

EMPNO	ENAME	PROJECTID
5001	A	101
5002	B	100
5003	C	100
5004	D	101
5005	E	
5006	F	

PROJECTID	PNAME	DURATION
100	X	2
101	Y	5
102	Z	3

Equi Join

ename	pname

Left Outer Join

Right Outer Join

Full Outer Join

Sub Queries

Monday, September 11, 2023 6:15 PM

Sub Queries / Nested Queries:

Syntax:

```
SELECT <column_list>
FROM <table_name>
WHERE <column> <operator> (<SELECT query>);
```

- A query which is written in another query is called "Sub Query / Nested Query".
- Outside query is called "Outer Query / Main Query / Parent Query".
- Inside Query is called "Inner Query / Sub Query / Child Query".
- When we don't know filter condition value, to find it we write Sub Query.
- Sub query must be written in parenthesis.
- Sub Query must be SELECT QUERY only. It cannot be INSERT / UPDATE / DELETE. Because, sub query has to find some value. Only SELECT can find the value. INSERT / UPDATE / DELETE cannot find the value.
- Outer query can be SELECT / INSERT / UPDATE / DELETE.
- Normally, First inner query gets executed. Then outer query gets executed. Result of inner query becomes input for Outer Query.
- We can write 255 levels of sub queries in a WHERE clause.

Types of Sub Queries:

5 Types:

- **Single Row Sub Query**
- **Multi Row Sub Query**
- **Correlated Sub Query**
- **Inline View / Inline Sub Query**
- **Scalar Sub Query**

Single Row Sub Query:

- **If sub query returns 1 row then it is called "Single Row Sub Query".**

Examples on Single Row Sub Query:

Display the emp records whose job title is same as BLAKE [display all managers records]:

BLAKE	MANAGER

```
SELECT ename,job,sal
FROM emp
WHERE job=(Find BLAKE job title);

SELECT ename,job,sal
FROM emp
WHERE job=(SELECT job FROM emp
WHERE ename='BLAKE');
```

Find max salary in all emps:

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

Find 2nd max salary:

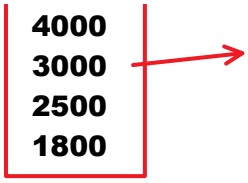
```
sal
-------
2000
4000
3000
5000
2500
1800
```

```
SELECT max(sal) FROM emp
WHERE sal<(Find max sal);
```

```
2000
4000
3000
```

→ max sal => 4000 => 2nd max sal

2500
1800



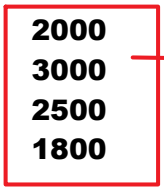
max sal => 4000 => 2nd max sal

```
SELECT max(sal) FROM emp
WHERE sal<(SELECT max(Sal) FROM emp);
```

Find 3rd max salary:

```
sal
-------
2000
4000
3000
5000
2500
1800
```

```
SELECT max(sal) FROM emp
WHERE sal<(Find 2nd max sal);
```



max sal => 3000 => 3rd max sal

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

to find 2nd max sal	write 1 sub query
to find 3rd max sal	write 2 sub queries
..	
to find 5th max sal	write 4 sub queries
..	
to find nth max sal	write n-1 sub queries

Find the emp name who is earning max salary:

max sal	5000

```
SELECT ename
```

FROM emp
WHERE sal=(Find max sal);

SELECT ename
FROM emp
WHERE sal=(SELECT max(sal) FROM emp);

Find the emp name who is earning 2nd max salary:

2nd max sal	3000

SELECT ename
FROM emp
WHERE sal=(Find 2nd max sal);

SELECT ename
FROM emp
WHERE sal = (SELECT max(sal) FROM emp
WHERE sal<(SELECT max(sal) FROM emp));

Assignment:
Find the emp name who is earning 3rd max sal
Find 2nd min salary
Find 3rd min salary

Display the emp records who are senior to BLAKE:

BLAKE hiredate	1-MAY-1981

SELECT ename,hiredate
FROM emp
WHERE hiredate<(Find BLAKE hiredate);

SELECT ename,hiredate
FROM emp
WHERE hiredate<(SELECT hiredate
FROM emp WHERE ename='BLAKE');

Assignment:

Display juniors of JAMES

Find most senior's name:

SMITH hiredate	17-DEC-1980

```
SELECT ename
FROM emp
WHERE hiredate=(Find most senior's hiredate);
```

```
SELECT ename
FROM emp
WHERE hiredate = (SELECT min(hiredate)
FROM emp);
```

Assignment:
Find most junior's emp name

Update 7900 emp sal as 30th dept max sal:

30th max sal	2850

```
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's record:

SMITH hiredate	17-DEC-1980

```
DELETE FROM emp
WHERE hiredate=(Find most senior's hiredate);
```

```
DELETE FROM emp
WHERE hiredate=(SELECT min(hiredate) FROM emp);
```

Find the deptno which is spending max amount on their emps:

max sum of sal	**10875**

```
SELECT deptno
FROM emp
GROUP BY deptno
HAVING sum(sal) = (Find max sum of sal in all depts);

SELECT deptno
FROM emp
GROUP BY deptno
HAVING sum(sal) = (SELECT max(sum(sal)) FROM emp
GROUP BY deptno);
```

Find the dept name which is spending max amount on their emps:

```
SELECT dname FROM dept
WHERE deptno = (find the deptno which is spending
max amount on their emps);

SELECT dname FROM dept
WHERE deptno = (SELECT deptno
FROM emp GROUP BY deptno
HAVING sum(sal) = (SELECT max(sum(sal)) FROM emp
GROUP BY deptno));
```

Assignment:

- **Find the deptno which is having max no of emps**
- **Find the dept name which is having max no of emps**

Note:

- **Relational Operators can be used to compare with one value.**
- **For single row sub query we can use relational operators. Because, sub query returns 1 value.**
- **For multi row sub query we cannot use relational operators. Because, sub query returns multiple values.**

Multi Row Sub Queries:

- **If sub query returns multiple rows then it is called "Multi Row Sub Query".**
- **For multi row sub query use following operators:**
  - **IN**
  - **ANY**
  - **ALL**

Example on multi row sub queries:

Display the emp records whose job title is same as SMITH and BLAKE job titles:
[Display all clerks and managers records]

SMITH	**CLERK**
BLAKE	**MANAGER**

```
SELECT ename,job,sal
FROM emp
WHERE job IN(find SMITH and BLAKE job titles);
```

```
SELECT ename,job,sal
FROM emp
WHERE job IN(SELECT job FROM emp
WHERE ename IN('SMITH','BLAKE'));
```

ALL:

- **It is used to compare column value with multiple values**
- **It avoids multi relational conditions using AND**

Note:

AND	**ALL conditions should be satisfied**

```
sal
--------
4000
2500
```

4000
2500
1800

Note:

AND	ALL conditions should be satisfied

Syntax:

WHERE <column> <relational_operator> ALL(value_list>)

Example:

WHERE sal>ALL(2000,3000) It checks sal value > all values in the list or not If sal > all values in list, condn => T otherwise, condn => F	WHERE sal>2000 AND sal>3000	
WHERE sal<ALL(2000,3000) It checks sal value < all values in the list or not If sal < all values in list, condn => T otherwise, condn => F	WHERE sal<2000 AND sal<3000	

OR	any 1 condition should be satisfied

ANY:

- It is used to compare column value with multiple values
- It avoids multi relational conditions using OR.

Syntax:

WHERE <column> <relational_operator> ANY(<value_list>)

Examples:

WHERE sal>ANY(2000,3000)	WHERE sal>2000 OR sal>3000	
WHERE sal=ANY(2000,3000)	WHERE sal=2000 OR sal=3000	WHERE sal IN(2000,3000)

=ANY	IN

IN:

- It is used to compare column value with multiple values.
- It avoids multi equality condition using OR

Example:

WHERE sal IN(2000,3000)	WHERE sal=2000 OR sal=3000
WHERE job IN('CLERK','MANAGER')	WHERE job='CLERK' OR job='MANAGER'

Display the emp records who are earning more than all managers:

```
2975        SELECT ename,sal
2850        FROM emp
2450        WHERE sal>ALL(Find all managers salaries);
```

```
SELECT ename,sal
FROM emp
WHERE sal>ALL(SELECT sal FROM emp
WHERE job='MANAGER');
```

Display the emp records who are earning more than any of the managers:

```
2975        SELECT ename,sal
2850        FROM emp
2450        WHERE sal>ANY(find all managers salaries);
```

```
SELECT ename,sal
FROM emp
WHERE sal>ANY(SELECT sal FROM emp
WHERE job='MANAGER');
```

Inline View / Inline Sub Query:

- **If sub query is written in FROM clause then it is called "Inline View / Inline Sub Query".**
- **no of sub queries in FROM clause is unlimited.**

- **Sub Query acts like table.**
- **To control the execution order of clauses, we need to write sub query in FROM clause**

Syntax:

```
SELECT <column_list>
FROM (<SELECT query>)
WHERE <condition>;
```

Execution Order

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

Examples on Inline View:

Find 3rd max salary (or) Display 3rd max salaried emp record:

```
SELECT ename,sal,
dense_rank() over(order by sal desc) as rnk
FROM emp
WHERE rnk=3;
```

Output:

ERROR: rnk invalid identifier

We cannot use column alias in WHERE.

Because WHERE gets executed before SELECT.

```
SELECT *
FROM (SELECT ename,sal,
dense_rank() over(order by sal desc) as rnk
FROM emp)
WHERE rnk=3;
```

*	All columns of sub query ename,sal,rnk

```
FROM
WHERE
SELECT
```

Find 5th max sal:

```
SELECT *
FROM (SELECT ename,sal,
dense_rank() over(order by sal desc) as rnk
FROM emp)
WHERE rnk=5;
```

Find nth max sal:

```
SELECT *
FROM (SELECT ename,sal,
dense_rank() over(order by sal desc) as rnk
FROM emp)
WHERE rnk=&n;
enter value for n: 1
max sal

/
enter value for n: 2
2nd max sal

enter value for n: 10
10th max sal
```

Display top 3 salaried emp records:

max sal	1
2nd max sal	2
3rd max sal	3

WHERE rnk<=3

```
SELECT *
FROM (SELECT ename,sal,
dense_rank() over(order by sal desc) as rnk
FROM emp)
WHERE rnk<=3;
```

Display top 5 salaried emp records:

```
SELECT *
FROM (SELECT ename,sal,
dense_rank() over(order by sal desc) as rnk
FROM emp)
WHERE rnk<=5;
```

Display top n salaried emp records:

```
SELECT *
FROM (SELECT ename,sal,
dense_rank() over(order by sal desc) as rnk
FROM emp)
WHERE rnk<=&n;
```

Pseudo Columns:

- **Pseudo => False**
- **Pseudo Column means, it looks like column of table. But, it is not column.**

Examples:

- **ROWNUM**
- **ROWID**

ROWNUM:

- **It is a pseudo column.**
- **it is used to get row number.**
- **this row number will be applied on result of SELECT query.**

Examples:

Apply row numbers to all emp table records:

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

Apply row numbers to all managers records:

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

ROWID:

- **It is a pseudo column.**
- **It is used to get row address.**

Example:

Display rowids of all emp table records:

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

Display 3rd row in emp table:

```
SELECT *
FROM (SELECT rownum AS rn, empno, ename, sal
FROM emp)
WHERE rn=3;
```

Display 1st, 5th and 12th rows from emp table:

```
SELECT *
FROM (SELECT rownum AS rn, empno, ename, sal
FROM emp)
WHERE rn IN(1,5,12);
```

Display even numbered rows from emp table:

```
SELECT *
FROM (SELECT rownum as rn,empno,ename,sal
FROM emp)
WHERE mod(rn,2)=0;
```

Display the row numbers between 5 to 10 from emp table:

```
SELECT *
FROM (SELECT rownum AS rn, empno, ename, sal
FROM emp)
WHERE rn BETWEEN 5 AND 10;
```

Note:
ROWID can be used to delete the duplicate records

Example on ROWID:

STUDENT

sid	sname	scity
1	A	HYD
1	A	HYD
2	B	DLH

```
DELETE FROM student
WHERE  rowid='<rowid>';

DELETE FROM student
WHERE rowid='AAATYaAAHAAAALXAAA';
```

Scalar Sub Query:

- **If sub query is written in SELECT clause then it is called "Scalar Sub Query".**
- **In SELECT clause we can write unlimited no of sub queries**
- **In this, sub query acts like column.**

```
WHERE => 255 sub queries
FROM   => unlimited
SELECT => unlimited
```

Examples on Scalar Sub Query:

Display no of records in emp and dept tables:

```
SELECT (SELECT count(*) FROM emp) AS emp,
(SELECT count(*) FROM dept) AS dept
FROM dual;
```

Output:

```
EMP         DEPT
---------   -------
14          4
```

Calculate share of each dept in salaries:

deptno	sum_of_sal	tot_amount	per
10	10000	50000	20
20	15000	50000	30

```
SELECT deptno, sum(Sal) AS sum_of_sal,
(SELECT sum(sal) FROM emp) AS tot_amount,
TRUNC(sum(sal)*100/(SELECT sum(Sal) FROM emp),2) as per
FROM emp
GROUP BY deptno
ORDER BY deptno;
```

Non-Correlated Sub Queries:

- single row sub query
- multi row sub query
- inline view
- scalar

In Non-Correlated Sub Query,

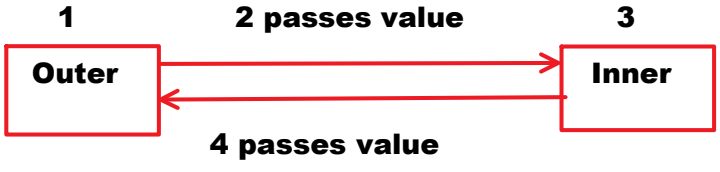
- First Inner query gets executed
- Then outer query gets executed
- Inner query gets executed only one time

Correlated Sub Query:

- If Outer query passes value to inner query then it is called "Correlated Sub Query".

- In Correlated Sub Query,
  - First outer query executed
  - Then inner query gets executed
  - Inner query gets executed for multiple times

Execution Process of Correlated Sub Query:



5 condn => T selects row

1. **Outer query gets executed. it selects a row.**
2. **Outer query passes value to inner query**
3. **Inner query gets executed.**
4. **Inner query passes value to outer query.**
5. **Outer query condition will be tested. If condition is TRUE, selects the row**
6. **These above 5 steps will be executed repeatedly for every row selected by outer query**

Note:

- **In Correlated Sub Query Inner Query gets executed multiple times**
- **no of execution times of inner query = no of rows selected by outer query**

Example on Correlated Sub Query:

Display the emp records who are earning more than their dept's average salary:

```
SELECT empno,ename,deptno,sal
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	10	15000
1003	C	20	25000
1004	D	20	5000

deptno	avg_sal
10	12500
20	15000

Output:

1002	B	10	15000
1003	C	20	25000

Sub Query => a query which is written in another query

```
SELECT
....        (SELECT
```

Types:

- **Non-Correlated => Inner [1time]=> outer**
  - **Single Row Sub Query**
  - **Multi Row**
  - **Inline**
  - **Scalar**
- **Correlated => Outer => Inner [multiple times]**

Display the emp records whose salary is more than BLAKE:

BLAKE	**2850**

```
SELECT ename,sal
FROM emp
WHERE sal>(Find BLAKE sal);
```

```
SELECT ename,sal
FROM emp
WHERE sal>(SELECT sal FROM emp WHERE ename='BLAKE');
```

Constraints

Thursday, September 14, 2023 6:21 PM

Constraints in SQL:

Constraint:

- **Constraint => Restrict / Control / Limit**

max marks: 100
0 TO 100

gender	m1
M	78
F	67
Z => ERROR	123 => ERROR

- **Constraint is a rule that is applied on a column.**
- **It restricts the user from entering invalid data.**
- **With this, we can maintain accurate and quality data.**
- **Maintaining accurate and quality data is called "Data Integrity".**
- **To implement data integrity we use constraints.**

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**

Example:

EMPLOYEE

PK

EMPNO	ENAME	JOB	SAL
1001	A	CLERK	6000
1002	B	ANALYST	6000
1003	A	ANALYST	4000
	C	MANAGER	10000
1001	D	MANAGER	9000

null
ERROR:

duplicate
ERROR:

- **When value is mandatory and it should not be duplicated then use Primary Key.**

- **A table can have only one column as primary key.**

Example:

```
CREATE TABLE t1(f1 NUMBER(4) PRIMARY KEY);

INSERT INTO t1 VALUES(1001);
INSERT INTO t1 VALUES(1002);

INSERT INTO t1 VALUES(null);  --ERROR

INSERT INTO t1 VALUES(1001);  --ERROR
```

Not Null:

- **It does not accept nulls.**
- **It accepts duplicates.**
- **When value is mandatory and it can be duplicated then use Not Null.**

Example:

EMPLOYEE

NOT NULL

EMPNO	ENAME	SAL
1	A	5000
2	B	3000
3	A accepts it	3000
4		8000

ERROR:

Example:

```
CREATE TABLE t2(f1 NUMBER(4) NOT NULL);

INSERT INTO t2 VALUES(1001);
INSERT INTO t2 VALUES(1002);

INSERT INTO t2 VALUES(1001);    --accepts duplicate value

INSERT INTO t2 VALUES(null);    --ERROR
```

UNIQUE:

- **It does not accept duplicates**

- It accepts nulls
- When value is optional and should not be duplicated then use UNIQUE

Example:

CUSTOMER UNIQUE

cid	cname	mailid
1001	A	abcd@gmail.com
1002	B	
1003	C	xyz@gmail.com
1004	D	abcd@gmail.com
1005	E	
1006	F	
1007	G	pqr@gmail.com

accepts null

ERROR: duplicate

Example:

```
CREATE TABLE t3(f1 NUMBER(4) UNIQUE);

INSERT INTO t3 VALUES(1001);
INSERT INTO t3 VALUES(1001);    --ERROR

INSERT INTO t3 VALUES(null);  --accepts it
INSERT INTO t3 VALUES(null);  --accepts it
```

Constraint	Duplicate	Null
Primary Key	NO	NO
Not Null	YES	NO
Unique	NO	YES

Primary Key = Not Null + Unique

CHECK:

- It is used to apply our own condition [rule] on column

Example:

max marks: 100
0 TO 100

STUDENT CHECK(m1 BETWEEN 0 AND 100)

sid	sname	m1
1001	A	55
1002	B	78
1003	C	564 ERROR
1004	D	-12 ERROR

DEFAULT:

- **It is used to apply default value to column**

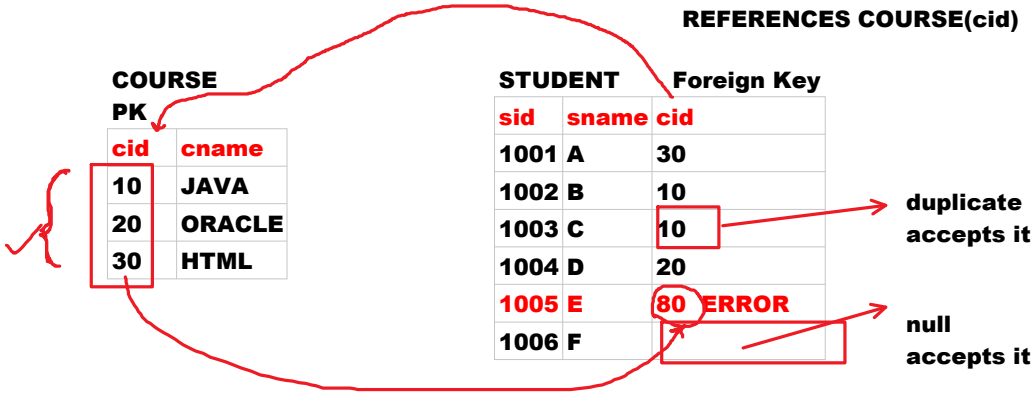
Example:

STUDENT DEFAULT 20000

sid	sname	FEE
1001	A	20000
1002	B	20000
1003	C	20000
1004	D	10000 accepts it

int x;

REFERENCES [FOREIGN KEY]:



- **Foreign Key can accept Primary Key values of another table.**
- **It can accept duplicates**
- **It can accept nulls**
- **It is used to establish relationship between 2 tables.**

Primary Key	**No duplicates, No nulls**
Not Nulls	**No nulls**
Unique	**No duplicates**
Check	**to apply our own condition**
Default	**to apply default value to column**
References {Foreign Key}	**can accept PK values of another table**

Examples on Constraints:

user_list

user_id	uname	pwd

user_id	don't accept duplicates and nulls	Primary Key
uname	don't accept duplicates and nulls	Unique Not Null
pwd	must have min 8 chars	Check

```
CREATE TABLE user_list
(
user_id NUMBER(4) PRIMARY KEY,
uname VARCHAR2(20) UNIQUE NOT NULL,
pwd VARCHAR2(20) CHECK(length(pwd)>=8)
);

INSERT INTO user_list VALUES(1001,'abcd','abcd12345');

INSERT INTO user_list VALUES(1001,'xyz','abcdefghij');
Output: ERROR: 1001 duplicated=> PK does not accept duplicate

INSERT INTO user_list VALUES(1002,'raju','kumar');
Output: ERROR: password must have min 8 chars
```

Primary Key	no dup, no nulls
Not Null	no nulls
Unique	no duplicates
Check	apply our own condn
Default	to apply default value
References [FK]	to accept PK values of another table

Example:

EMPLOYEE

empno	ename	gender	sal	mail_id

empno	don't accept duplicates and nulls	Primary Key
ename	don't accept nulls	Not Null
gender	accept M or F	CHECK
sal	min 7000	CHECK
mail_id	don't accept duplicates and nulls	Unique Not Null

```
CREATE TABLE employee
(
empno NUMBER(4) PRIMARY KEY,
ename VARCHAR2(10) NOT NULL,
gender CHAR CHECK(gender IN('M','F')),
sal NUMBER(8,2) CHECK(sal>=7000),
mail_id VARCHAR2(30) UNIQUE NOT NULL
);
```

Example:

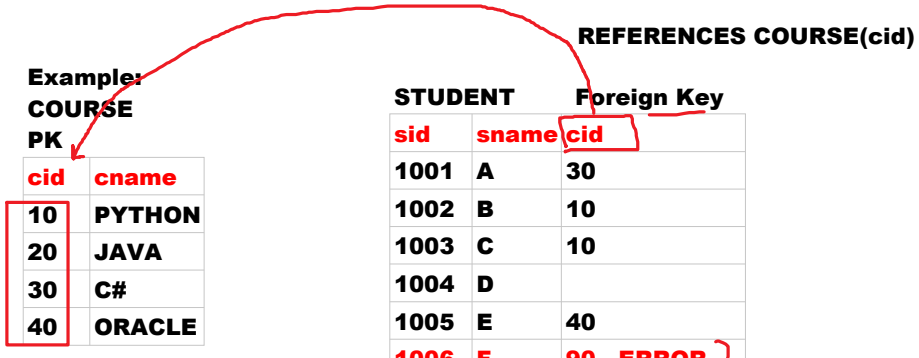
STUDENT

PK	Not Null	default 'NARESH'	default 'HYD'	default 20000
sid	sname	cname	ccity	fee
1	A	NARESH	HYD	20000
2	B	NARESH	HYD	20000
3	C	NARESH	HYD	20000

```
CREATE TABLE student
(
sid NUMBER(4) PRIMARY KEY,
sname VARCHAR2(10) NOT NULL,
cname VARCHAR2(10) DEFAULT 'NARESH',
ccity VARCHAR2(3) DEFAULT 'HYD',
fee NUMBER(7,2) DEFAULT 20000
);

INSERT INTO student VALUES(1,'A');
Output:
ERROR: not enough values

INSERT INTO student(sid,sname) VALUES(1,'A');
```



PK	FK	
cid NUMBER(2)	cid NUMBER(2)	valid
	cid DATE => ERROR	invalid

- **Foreign Key column data type should be matched with Primary Key column data type**

- **PK column name and FK column name need not to be same.**

PK	FK	
cid	courseid	valid

```
CREATE TABLE course
(
cid NUMBER(2) PRIMARY KEY,
cname VARCHAR2(10)
);

CREATE TABLE student
(
sid NUMBER(4),
sname VARCHAR2(10),
cid NUMBER(2) REFERENCES COURSE(cid)
);
```

Naming Constraints:

- We can give names to constraints.
- To identify constraint uniquely we can define a name.
- To drop the constraint or enable the constraint or disable the constraint this constraint will be used.
- As a developer, when we define a constraint it is better to define a name. If we don't define constraint name implicitly ORACLE defines constraint name.
- ORACLE defined name is:
  a 6 digit random number will be prefixed with sys
  Example: SYS_C006712

- "CONSTRAINT" keyword is used to give constraint name.

Example on naming constraints:

STUDENT

sid	sname	m1

sid	don't accept duplicates and nulls	PRIMARY KEY	c1
sname	don't accept nulls	NOT NULL	c2
m1	marks must be b/w 0 to 100	CHECK	c3

```
CREATE TABLE student
(
sid NUMBER(4) CONSTRAINT c1 PRIMARY KEY,
sname VARCHAR2(10) CONSTRAINT c2 NOT NULL,
m1 NUMBER(3) CONSTRAINT c3 CHECK(m1 BETWEEN 0 AND 100)
);
```

USER_CONSTRAINTS:

- It is a system table / built-in table

- **It maintains all constraints information of a user**

DESC user_constraints;

to see constraints information:

```
COLUMN table_name FORMAT A15

SELECT table_name, constraint_name, constraint_type
FROM user_constraints;

SELECT table_name, CONSTRAINT_NAME, CONSTRAINT_TYPE
FROM user_constraints
WHERE table_name='STUDENT';
```

Output:

```
TABLE_NAME     CONSTRAINT_N   C
-------------  ------------------------  ----
STUDENT          C2                 C
STUDENT          C3                 C
STUDENT          C1                 P
```

Note:
We cannot give constraint name to "DEFAULT"

We can apply a 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.**

Example:

```
CREATE TABLE student
(
sid NUMBER(4) CONSTRAINT c1 PRIMARY KEY,
sname VARCHAR2(10) CONSTRAINT c2 NOT NULL,
m1 NUMBER(3) CONSTRAINT c3 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.**
  [PRIMARY KEY, UNIQUE, CHECK, REFERENCES]
- **Not Null and Default constraints cannot be applied at table level.**

```
CREATE TABLE student1
(
sid NUMBER(4),
sname VARCHAR2(10) CONSTRAINT c5 NOT NULL,
m1 NUMBER(3),
CONSTRAINT c4 PRIMARY KEY(sid),
CONSTRAINT c6 CHECK(m1 BETWEEN 0 AND 100)
);
```

Example:

Applying PK and FK at table level:

REFERENCES DEPT1(DEPTNO)

DEPT1
PK
c11

DEPTNO	DNAME
10	SALES
20	HR
30	RESEARCH

EMP1
PK **FK**
c12 **c13**

EMPNO	ENAME	DEPTNO
1001	A	30
1002	B	10
1003	C	90 ERROR

```
CREATE TABLE dept1
(
deptno NUMBER(2),
dname VARCHAR2(10),
CONSTRAINT c11 PRIMARY KEY(deptno)
);
```

```
CREATE TABLE emp1
(
empno NUMBER(4),
ename VARCHAR2(10),
deptno NUMBER(2),
CONSTRAINT c12 PRIMARY KEY(empno),
CONSTRAINT c13 FOREIGN KEY(deptno) REFERENCES dept1(deptno)
);
```

Why to use table level?

Table Level Constraint is mainly used for 2 reasons. they are:

- **to apply combination of columns as primary key or unique [composite key]**
- **to use another column name in constraint**

Example:

STUDENT **PK(sid,subject)**

sid	sname	subject	marks
1001	A	M1	70
1001	A	M2	90
1001	A	M3	55
1002	B	M1	88
1002	B	M2	55
1002	B	M3	70
1001	A	M1	67
1002	B	M2	23
	C	M1	88
1003	C		80

ERROR=> duplicate record
ERROR=>duplicate record
ERROR => sid is null
ERROR => subject is null

- If we set combination of columns as primary key then it is called "Composite Primary Key"
- Composite primary key can set at table level only.

STUDENT

sid	sname	subject	marks

PK(sid,subject) => Composite Primary Key

```
CREATE TABLE student
(
sid NUMBER(4),
sname VARCHAR2(10),
subject CHAR(2),
marks NUMBER(3),
CONSTRAINT c15 PRIMARY KEY(sid,subject)
);
```

CM_LIST

CMID	state_code	CM_NAME	START_DATE	END_DATE
101	TS	KCR	23-MAY-2019	23-MAY-2014

```
CREATE TABLE cm_list
(
cmid NUMBER(3),
state_code CHAR(2),
cm_name VARCHAR2(10),
start_date DATE,
end_date DATE CHECK(end_date>start_date)
);
```

Output:
ERROR:

```
CREATE TABLE cm_list
(
cmid NUMBER(3),
```

```
state_code CHAR(2),
cm_name VARCHAR2(10),
start_date DATE,
end_date DATE,
CONSTRAINT c20 CHECK(end_date>start_date)
);
```

PRODUCTS

PID	PNAME	MANUFATUREDDATE	EXPIRYDATE
1234	AA	16-SEP-2023	16-DEC-2020

Adding, Dropping, enabling, disabling, renaming constraints:

ALTER:
Using ALTER command we can:

- **Add the constraints => ADD CONSTRAINT**
- **rename the constraints => RENAME CONSTRAINT**
- **disable the constraints => DISABLE CONSTRAINT**
- **enable the constraints => ENABLE CONSTRAINT**
- **drop the constraints => DROP CONSTRAINT**

Example:

STUDENT

sid	sname	m1

Student is existing table.
Add PK to sid
Add Check constraint to m1
Add not null to sname

Creating table:

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

Adding Primary Key:

```
ALTER TABLE student ADD CONSTRAINT c21 PRIMARY KEY(sid);
```

Note:
USING ADD keyword we can add Table Level Constraints only
USING MODIFY keyword we can add Column Level Constraints.

Adding CHECK:

```
ALTER TABLE student ADD CONSTRAINT c22 CHECK(m1 BETWEEN 0 AND 100);
```

Adding NOT NULL:

```
ALTER TABLE student MODIFY sname not null;
```

Renaming Constraint:

sid	**PK**	**c21 => Z**

ALTER TABLE student RENAME CONSTRAINT c21 TO z;

Disabling Primary Key:

ALTER TABLE student
DISABLE CONSTRAINT z;

z	**PK**	**temporarily PK will not work when we disable it**

Enabling Primary Key:

ALTER TABLE student
ENABLE CONSTRAINT z;

Note:
To enable PK constraint, it should not contain duplicates and nulls.

Dropping Primary Key:

ALTER TABLE student
DROP CONSTRAINT z;

Syntax of creating table:

```
CREATE TABLE <table_name>
(
<field_name> <data_type> [CONSTRAINT <con_name> <con_type>,
<field_name> <data_type> CONSTRAINT <con_name> <con_type>,
.
.]
);
```

Field Definition:

```
<field_name> <data_type> CONSTRAINT <con_name> <con_type>
```

constraints

views
sequences
mviews

Wednesday
set operators => tables

PL/SQL

SET OPERATORS

Wednesday, September 20, 2023 6:15 PM

A = {1,2,3,4,5}

B = {4,5,6,7,8}

A U B = {1,2,3,4,5,6,7,8} = B U A

A UA B = {1,2,3,4,5,4,5,6,7,8} = B UA A

A I B = {4,5} = B I A

A M B = {1,2,3} => specific elements of A

B M A = {6,7,8}

SET OPERATORS:

- **SET OPERATOR is used to combine result of 2 SELECT queries.**

  Syntax:

  SELECT query
  SET OPERATOR
  SELECT query;

ORACLE SQL provides following SET operators:

- **UNION**
- **UNION ALL**
- **INTERSECT**
- **MINUS**

CUSTOMER1 [branch-1]

cid	cname
1001	A
1002	B
1003	C

CUSTOMER2 [branch-2]

cid	cname
5001	D
1002	B
5002	E

UNION:

- **It combines result of 2 SELECT queries without duplicates.**

Display all customers of branch-1 and branch-2:

```
SELECT cid,cname FROM customer1
UNION
SELECT cid,cname FROM customer2;
```

1001	A
1002	B
1003	C
5001	D
5002	E

UNION ALL:

- **It combines result of 2 SELECT queries including duplicates.**

Display all customers of branch-1 and branch-2 including duplicates:

```
SELECT cid,cname FROM customer1
UNION ALL
SELECT cid,cname FROM customer2;
```

1001	A
1002	B
1003	C
5001	D
1002	B
5002	E

What are the differences between UNION and UNION ALL?

UNION	UNION ALL
It does not give duplicate records.	It can give duplicate records
gives result in order.	does not give result in order.
Slower	Faster

Intersect:

- **used to common records from the result of 2 select queries**

Example:

Display common customers of branch1 and branch2:

```
SELECT cid,cname FROM customer1
INTERSECT
SELECT cid,cname FROM customer2;
```

1002	B

MINUS:
it is used get specific records from first select query result.

Example:
Display specific customers of branch-1:

```
SELECT cid,cname FROM customer1
MINUS
SELECT cid,cname FROM customer2;
```

cid	cname
1001	A
1003	C

Display specific customers of branch-2:

```
SELECT cid,cname FROM customer2
MINUS
SELECT cid,cname FROM customer1;
```

cid	cname
5001	D
5002	E

Display the job titles offered by deptno 10 and 20:

10th dept job titles

MANAGER
PRESIDENT
CLERK

20th dept job titles

CLERK
MANAGER
ANALYST
CLERK
ANALYST

```
SELECT job FROM emp WHERE deptno=10
UNION
SELECT job FROM emp WHERE deptno=20;
```

Output:

```
JOB
---------
MANAGER
PRESIDENT
CLERK
ANALYST
```

Display common job titles offered by deptno 10 and 20:

```
SELECT job FROM emp WHERE deptno=10
INTERSECT
SELECT job FROM emp WHERE deptno=20;
```

```
JOB
---------
MANAGER
CLERK
```

Display specific job titles offered by deptno 10:

```
SELECT job FROM emp WHERE deptno=10
MINUS
SELECT job FROM emp WHERE deptno=20;
```

```
PRESIDENT
```

Display specific job titles offered by deptno 20:

```
SELECT job FROM emp WHERE deptno=20
MINUS
SELECT job FROM emp WHERE deptno=10;

job
------
ANALYST
```

Rules:

- **Data Types of Corresponding columns in both SELECT queries must be same.**

Example:

```
SELECT cid,cname FROM customer1
UNION
SELECT cname,cid FROM customer2;
```

Output:
ERROR:

- **No of Columns in both SELECT queries must be same**

Example:

```
SELECT cid FROM customer1
UNION
SELECT cid,cname FROM customer2;
```

Output:
ERROR

What is the difference between Joins and Set Operators:

JOINS	• combines the columns • vertical merging
UNION	• combines the rows • horizontal merging

ename	sal	dname	loc

1001	A
1002	B
1003	C
5001	D
5003	E

SQL:

- **Non-Procedural Language [no programs]**
- **we write queries**

SQL provides 5 sub languages:

DDL [metadata]	DRL / DQL [retrievals]	TCL [transactions]	DML [manipulations]	DCL [accessibility]
create alter drop flashbak purge truncate rename	select	commit rollback savepoint	insert update delete insert all merge	grant revoke

Built-In Functions:

String	lower(), upper(), initcap() RPAD() LPAD() RTRIM() LTRIM() TRIM()
Conversion	to_char() to_date() to_number()
Aggregate (or) Group	max() min() count() sum() avg()
Date	add_months() months_between() last_day() next_day() sysdate systimestamp
Number	trunc() round() mod() ceil() floor()
Analytic	rank() dense_rank() row_number()
Miscellaneous	NVL() NVL2()

Clauses:

FROM	**FROM emp**
WHERE	**WHERE sa>3000**
GROUP BY	**GROUP BY deptno**
HAVING	**HAVING sum(sal)>10000**
SELECT	**SELECT ename,sal**
DISTINCT	**SELECT DISTINCT job**
ORDER BY	**ORDER By sal DESC**

JOINS:

- **used to retrieve data from multiple tables**

Inner Join	**matched records only**	
	Equi-Join	**based on =**
	Non-Equi Join	**based on other than =**
Outer Join	**matched + unmatched records**	
	Left Outer	**matched + unmatched from L**
	Right Outer	**matched + unmatched from R**
	Full Outer	**matched + unmatched from L & R**
Self	**A table will be joined to itself**	
Cross	**each record in 1 table will be joined with every record in another**	

Sub Query:

A query written in another

Types:

single row sub query => sub query returns 1 row
multi row sub query => sub query returns >1 row
correlated => ouer => inner

inline view => writing sub query in FROM
scalar => writing sub query in SELECT

Constraints:

PRIMARY KEY
NOT NULL
UNIQUE
CHECK
DEFAULT
REFERENCES [FK]

SET:

UNION
UNION ALL
INTERSECT
MINUS