Saturday, September 14, 2024 9:02 AM

Admin:
akhil:
9154 1561 92 - whatsapp only

Notes link:
bit.ly/oracledbnotes

oracle software link:
bit.ly/oracle21csoftware

oracle software installation video link:
bit.ly/oracle21cinstallation

Oracle SQL & PLSQL @ 730 AM IST by Mr Shiva Chaitanya
Day-1 https://youtu.be/mHeeVzEZ5ZM
Day-2 https://youtu.be/oGF4woKC84Q
Day-3 https://youtu.be/ODwWaRzcZuc
Day-4 https://youtu.be/Q2czIx7vJLY
Day-5 https://youtu.be/apIBiC1xxmg
Day-6 https://youtu.be/tKgN667jgAA
Day-7 https://youtu.be/KK3N2ZV9510

SYLLABUS

Monday, September 16, 2024 7:55 AM

ORACLE

SQL

TABLES			
	SQL COMMANDS	**DDL, DRL, DML, TCL, DCL**	
	Built-In Functions	**String functions** **date functions** **analytic functions** **aggregate functions** . .	
	Clauses	**Group By** **Having** **Order By** . .	
	Joins	**Inner Join** **Equi Join** **Non-Equi join** **Outer Join** **Left outer join** **Right outer join** **Full outer join** **self -join** **cross join**	
	Sub Queries	**Non-correlated sub query** **single row sub query** **multi row sub query** **inline view** **scalar sub query** **Correlated sub query**	

	Constraints	**Primary key** **Foreign Key** **.** **.**	
	Set Operators	**UNION** **UNION ALL** **INTERSECT** **MINUS**	
VIEWS	**Types of views** **simple view** **complex view**		
INDEXES	**Types of Indexes** **B-tree Index** **simple** **composite** **unique** **function based** **Bitmap index**		
Materialized Views			
Sequences			
Synonyms			

PL/SQL

PL/SQL Basics	**data types** **declare** **assign** **print** **read**	
Control Structures	**IF .. THEN** **IF .. THEN .. ELSE** **.** **.** **FOR** **WHILE** **SIMPLE LOOP** **.**	

	▪	
CURSORS		
COLLECTIONS		
EXCEPTION HANDLING		
STORED PROCEDURES		
STORED FUNCTIONS		
PACKAGES		
TRIGGERS		
WORKING WITH LOBs		
DYNAMIC SQL		

ORACLE

Friday, September 13, 2024 7:47 AM

Data Store
Database
DBMS
RDBMS
Metadata

DATA

BANK

Run the business
Analyze the business

Branches
Customers
Transactions
Employees
.
.

Run the business

opening account
deposit
withdraw
closing account

Analyze the Business

2023 ?
2024 ?

AMAZON

Products
Customers
Orders
Payments
Wishlist
Sellers

Run the business

provides products info
placing order
payments

Analyze the business

2022 ?
2023 ?
2024 ?

GOAL:
storing business data permanently in computer

large amounts of business data

Storing data:

Storing data:

- Variable
- Object
- File
- Database

large amounts of business data
multiple users
secured

2020
2021
.
.
2024

Variable:
Variable is temporary.

int empid=1234;

empid
1234

Object:
Object is temporary.

Employee e1 = new Employee(1234, "Raju", 5000);

e1

EMPID	ENAME	SAL
1234	Raju	5000

File:

- File is permanent.

- File is suitable to store small amounts of data.
- it is developed for 1 user.
- less security.

Database:

- Database is permanent.

- It is suitable to store large amounts of data.
- it is developed for multiple users.
- It is secured.

searching for products
adding items to wishlist
placing order
payments

AMAZON DB SERVER

products
wishlist
orders
payments

Data Store:

- **data Store is a location where data is stored.**

Examples:

BOOK, FILE, DATABASE

before 1960s =>
computers were not available to store business data

Database:

- **DATABASE is a kind of data store.**

- **DATABASE is a location where organization's business data stored permanently.**

Examples:

AMAZON DB

products
customers
orders
payments

BANK DB

Branches
Customers
Transactions
Products

COLLEGE DB

courses
students
marks
fee

DATABASE contains interrelated data in an organized form.

interrelated =>
COLLEGE DB contains college related data. not bank related data.

organized form => arranged in meaningful way

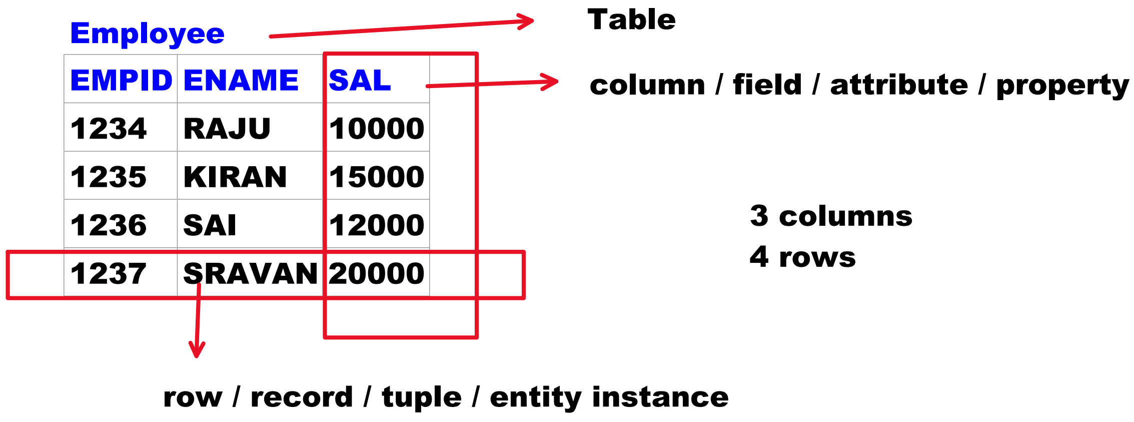
TABLE

SAL	ENAME	EMPID
6000	raju	1234

TABLE:

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

Example:



Employee

EMPID	ENAME	SAL
1234	RAJU	10000
1235	KIRAN	15000
1236	SAI	12000
1237	SRAVAN	20000

Table

column / field / attribute / property

3 columns
4 rows

row / record / tuple / entity instance

Saturday, September 14, 2024 7:43 AM

BANK DB

Branches
Customers
Transactions
Employees

Goal:
storing bus data permanently

DBMS:

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

Evolution of DBMS:

Before 1960s	**Books**
In 1960s	**FMS => File Management Software**
In 1970s	**HDBMS => Hierarchical DBMS** **NDBMS => Network DBMS**
In 1976	**E.F Codd => RDBMS concept**
In 1977	**ORACLE company founder: Larry Ellison** **estd a company: Software Development Inc.**
In 1979	**renamed company name: Relational Software Inc.** **introduced ORACLE software.**
In 1983	**renamed company name: ORACLE carp.**

RDBMS:

- **RDBMS => Relational DataBase Management System / Software.**
- **Relation => Table**

BANK DB

Branches table

IFSC_CODE	CITY	STATE	COUNTRY
IOB001234	HYD	TG	INDIA

Customers

CID	CNAME	AADHAR	PAN	MAILID	MOBILE

Transactions

TRANS_ID	T_TYPE	T_DATE_TIME	AMOUNT	ACNO

EMPLOYEE

EMPID	ENAME	JOB	SAL	DOJ

Examples of RDBMS:

ORACLE, SQL SERVER, DB2, POSTGRE SQL, MYSQL

ORACLE	oracle company
SQL SERVER	microsoft
DB2	IBM
POSTGRE SQL	Postgre Forum
MY SQL	sun micro systems

Laptop:
dell, lenovo, apple, microsoft

Metadata:

- **Metadata is the data about the data.**
- **Metadata can be also called as "Data Definition".**
- **Examples:**
  column name, table name, data type, column size

EMPLOYEE → metadata

EMPID NUMBER(4) -9999 TO 9999	ENAME	SALARY
1234	RAJU	6000
1235	KIRAN	10000
RAJU => error		
25-DEC-2023 => error		
56789 => error		

metadata

data

Data Store	is a location => where data is stored Examples: BOOK, FILE, DATABASE
Database	is a kind of data store is a location where org bus data stored permanently

DBMS	is a software
	is used to create and maintain the database
	
	types of DBMSs:
	HDBMS => hierarchy [levels] => failed
	NDBMS => network => failed
	RDBMS => relation [table] => successful concept

RDBMS	is a software is used to create and maintain the database in the form of tables Examples: ORACLE, SQL SERVER, DB2, POSTGRE SQL, MY SQL
Metadata	is the data about the data data definition Examples: table name, column name, data type, column size

ORACLE

Saturday, September 14, 2024 8:48 AM

ORACLE:

- ORACLE is a relational database management software [RDBMS].

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

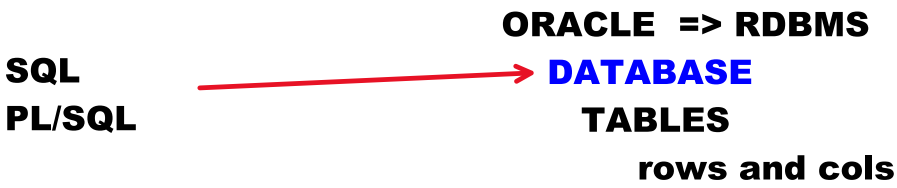
- Using ORACLE software we can store, manipulate and retrieve the data of database.

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

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

  retrieve => getting back => opening existing data

  checking balance
  Transaction statement
  searching for products



ORACLE => RDBMS

SQL

PL/SQL

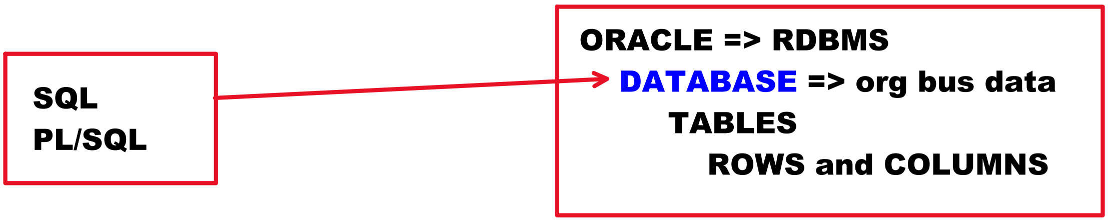
DATABASE

TABLES

rows and cols

Monday, September 16, 2024 7:45 AM

Data Store	**is a location => data is stored Examples: BOOK, FILE, DATABASE**
Database	**is a kind of data store is a location => org bus data stored permanently**
DBMS	**is a software is used to create and maintain the database**
RDBMS	**is a kind of DBMS is a software is used to create and maintain the database in the form of tables Relation => table Examples: ORACLE, SQL SERVER, DB2**
Metadata	**is the data about the data Data Definition Examples: column name, table name, data type**



SQL
PL/SQL

ORACLE => RDBMS
DATABASE => org bus data
TABLES
ROWS and COLUMNS

ORACLE

Monday, September 16, 2024 8:17 AM

ORACLE:

- **ORACLE is a relational database management software [RDBMS].**
- **Using it, we can store, manipulate and retrieve the data of database.**

manipulate => 3 actions => insert / update / delete

emp joined => insert
emp sal increased => update
emp resigned => delete

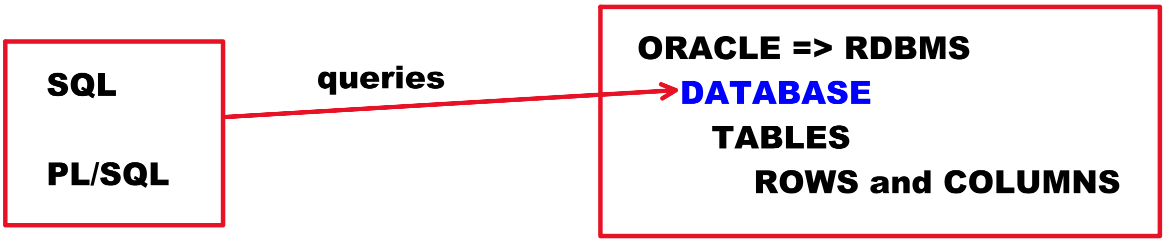
customer opening account => insert
withdraw => update
closing account: delete

retrieve => opening existing data

checking balance
transaction statement

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

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



SQL

PL/SQL

queries

ORACLE => RDBMS
DATABASE
TABLES
ROWS and COLUMNS

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

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

SQL:

- **SQL => Structured Query Language.**
- **SQL is a query language.**
- **In SQL, we develop the queries.**
- **These queries are developed to communicate with ORACLE DB.**
- **Query => is a request that is sent to DB SERVER.**

Example:

```
SELECT balance FROM accounts
WHERE acno=1234;   => query => request => DB SERVER
Output:
80000       response

SELECT ename, sal FROM emp;
```

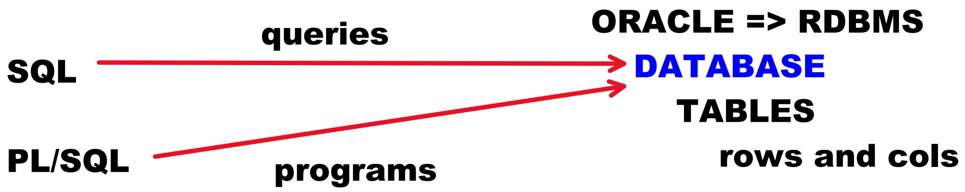
PL/SQL:

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

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

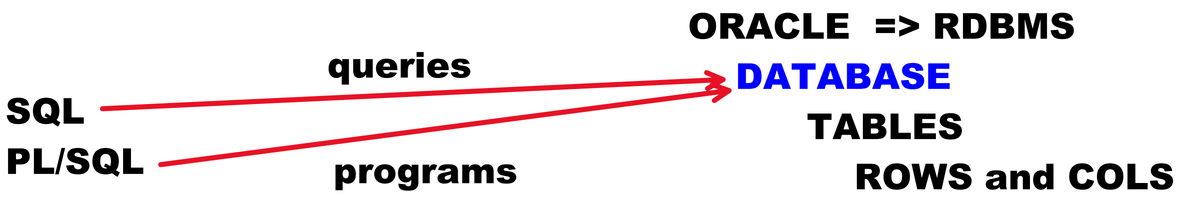
- **PL/SQL is extension of SQL.**

- **it is a programming language.**
- **in this we develop the programs to communicate with ORACLE DB.**



SQL

Tuesday, September 17, 2024 7:41 AM



ORACLE => RDBMS

queries

SQL

PL/SQL

programs

DATABASE

TABLES

ROWS and COLS

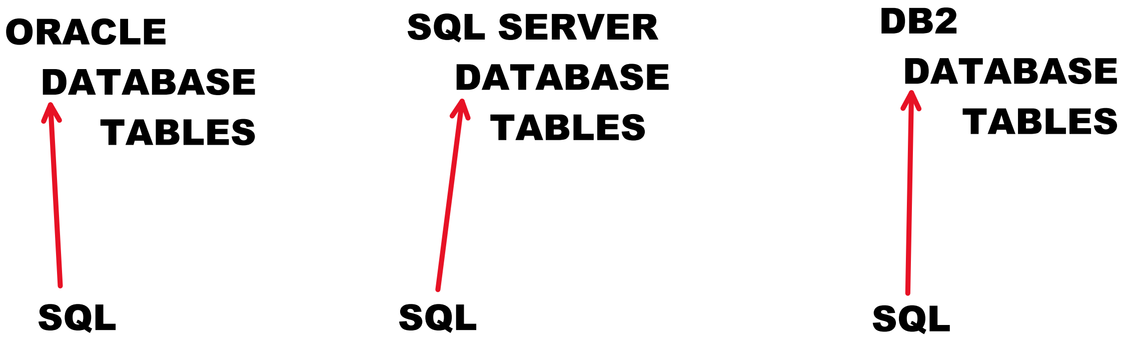
QUERY => request => DB SERVER

SQL:

- SQL => Structured Query Language.
- It is a query language.
- It is used to develop the queries.
- we develop queries to communicate with ORACLE DATABASE.
- Query => request / command / instruction
- Query is a request that is sent to DB SERVER.

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

- SQL is Unified Language. it is a common language to work with many relational databases.



ORACLE

DATABASE

TABLES

SQL

SQL SERVER

DATABASE

TABLES

SQL

DB2

DATABASE

TABLES

SQL

SQL Commands

Tuesday, September 17, 2024 8:29 AM

SQL Commands:

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 • **DRL => Data Retrieval Language** • **DQL => Data Query Language** • **retrieve => opening existing data** • **It deals with data retrievals**	**SELECT**

DML
• **DML => Data Manipulation Language**
• **It deals with data [data manipulations]**
• **manipulation => 3actions INSERT / UPDATE / DELETE**
	INSERT
	UPDATE
	DELETE
	INSERT ALL [Oracle 9i]
	MERGE [Oracle 9i]

TCL • **TCL => Transaction Control Language** • **It deals with transactions**	**COMMIT** **ROLLBACK** **SAVEPOINT**
DCL / ACL • **DCL => Data Control Language** • **ACL => Accessing Control Language** • **It deals with data accessibility**	**GRANT** **REVOKE**

CREATE

Tuesday, September 17, 2024 8:50 AM

ORACLE DB OBJECTS:

TABLE
VIEW
INDEX
SEQUENCE
SYNONYM
MATERIALIZED VIEW
PROCEDURE
FUNCTION
PACKAGE
TRIGGER

CREATE:

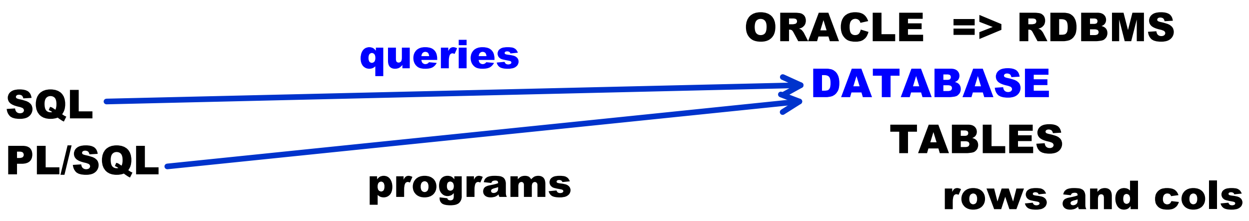
- CREATE command is used to create ORACLE DB OBJECTS like tables, views, indexes ...etc.

Syntax to create the table:

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

[ ]	Optional
< >	Any

Wednesday, September 18, 2024 7:42 AM



ORACLE => RDBMS

SQL → queries → DATABASE

PL/SQL → programs → DATABASE

TABLES

rows and cols

SQL commands:

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

CREATE

Wednesday, September 18, 2024 7:54 AM

CREATE:
It is used to create ORACLE DB OBJECTS
like tables, views, indexes .. etc.

Syntax:

```
CREATE TABLE  <name>
(
    <column> <data_type> ,
    <column> <data_type> ,
    .
);
```

EMPLOYEE

empid	ename	sal

Data Types in ORACLE SQL:

Data Type tells,

- **how much memory has to be allocated**
- **which type of should be accepted**

ORACLE SQL provides following data types:

Character Related
 	
Examples:
'RAJU'
'HYD'
 	
Note:
string => is a group of characters
	Char(n)
	Varchar2(n)
	CLOB
	Long
	
	nChar(n)
	nVarchar2(n)
	nCLOB

Integer Related **Examples:** **1234** **78** **18**	**Number(p)** **p => precision** **Integer** **Int**
Floating Point Related **Examples:** **12000.00** **2000.80** **67.89**	**Number(p, s)** **p => precision** **s => scale** **float** **binary_float** **binary_double**
Date and Time Related **Examples:** **17-AUG-23** **18-SEP-24 8.15.15.123456 AM** **Note:** **default date format: DD-MON-YY**	**Date** **Timestamp** **[oracle 9i]**
Binary Related **Examples:** **images, audios, videos**	**BFILE** **BLOB**

Character related data types:

- **These are used to hold strings.**
- **String => is a group of chars.**
- **String must be enclosed in single quotes.**
- **Examples: 'INDIA', 'HYD'**

Char(n)
Varchar2(n)
LONG
CLOB

nChar(n)
nVarchar2(n)
nCLOB

Char(n):

- **n => max no of chars**
- **It is fixed length data type.**
- **It is used to maintain fixed length chars.**
- **Extra memory will be filled with spaces.**
- **max size: 2000 Bytes [2000 chars]**
- **Default size: 1**

Varchar2(n):

- **n => max no of chars**
- **It is variable length data type.**
- **It is used to maintain variable length chars.**
- **Extra memory will not be filled with spaces.**
- **max size: 4000 Bytes [4000 chars]**
- **Default size: no default size**

```
Country_Code  CHAR(3)             ENAME       VARCHAR2(10)
---------------------             ------------
IND                               RAJU
AUS                               RAMESH
WIN
```

```
State_Code  CHAR(2)               PNAME    VARHCAR2(20)
-----------------                 --------------
TG                                keyboard
AP                                mouse
UP
```

Note:
All Character related data types can accept letters, digits and special chars.

```
vehicle_number  CHAR(10)          mailid    VARCHAR2(30)
------------------------          ---------------
TS09AA1234                        sai1234@gmail.com
                                  raj_kumar786@gmail.com
```

```
PAN_NUMBER  CHAR(10)              job   VARCHAR2(10)
------------------------          -------
ABCDE1234F                        MANAGER
```

CLERK

NOTE:

- **VARCHAR2 data type can hold maximum 4000 chars only.**
- **To hold more than 4000 chars we can use LONG or CLOB.**
- **It is better to use CLOB.**

LONG:

- **It is used to hold large amounts of chars.**
- **It has some restrictions:**
  - **A table can have only 1 column as LONG type.**
  - **we cannot use built-in functions on LONG type.**
- **max size: 2 GB**

CLOB:

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

Examples:

Feedback CLOB

Comments CLOB

Experience_summary CLOB

Product_features CLOB

Char(n)
Varchar2(n)
LONG
CLOB
	•**ASCII code char data types**
	•**Single Byte char data types**
	•**can hold english lang chars only**

nChar(n)
nVarchar2(n)
nCLOB
n => national
	•**UNI code char data types**
	•**Multi Byte char data types**
	•**can hold english lang chars and other lang chars also**

nChar(n)	•**n => no of chars** •**it is fixed length data type** •**max size: 2000 Bytes [1000 chars]**
nVarchar2(n)	•**it is variable length data type** •**max size: 4000 bytes [2000 chars]**

In C:

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

In Java:

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

ASCII:

- **American Standard Code for Information Interchange**
- **It is a coding system**
- **256 chars coded**
- **ASCII = english lang chars + digits + special chars**
- **ranges from 0 TO 255.**
- **255 => 1111 1111 => 8 bits [1 Byte]**

UNI:

- **UNIVERSAL**
- **Extension of ASCII**

	•max size: 2000 Bytes [1000 chars]
nVarchar2(n)	•it is variable length data type •max size: 4000 bytes [2000 chars]
nCLOB	•nvarchar2 can hold max 2000 chars only to hold more than 2000 chars we use nCLOB

UNI:

- UNIVERSAL
- Extension of ASCII
- UNI = ASCII + other lang chars
- 65536 chars coded
- ranges from 0 to 65535
- 65535 = 1111 1111 1111 1111 => 16 bits [2 Bytes]

Note:
To hold other language chars we use
national character set data types

Integer Related Data Types:

NUMBER(p):

- p => precision => max no of digits
- It is used to hold integers.
- p valid range => 1 to 38
- max size: 21 Bytes

Examples:

```
EMPID   NUMBER(4)            range: -9999 TO 9999
------------
1234
1235
567
56
5
9999
10000     => error
```

```
M1        NUMBER(3)            -999 TO 999
----------
78
100                                              Max Marks: 100
456
999
1000   ERROR
```

```
AGE    NUMBER(2)            -99 TO 99
------
23                                               18 to 30
99
100    ERROR
```

MOBILE_NUMBER   NUMBER(10)

AADHAR_NUMBER   NUMBER(12)

CREDIT_CARD_NUMBER NUMBER(16)

Integer and int:

integer = int = number(38)

- Integer and Int are alias names of number(38)

Floating point related data types:

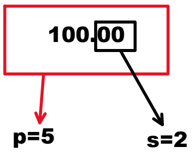
NUMBER(p,s):

- p => precision => max no of digits
- s => scale => max no of decimal places
- It is used to hold floating point values.

Examples:

-999.99 TO 999.99

```
AVRG_MARKS    NUMBER(5,2)
-----------------------
67.89
786.23
999.99
1000      => error
123.46789786868  => 123.47
123.46389786868  => 123.46
```


100.00

p=5

s=2

-999999.99 TO 999999.99

```
SALARY NUMBER(8,2)
------------
30000.00
700000.00
1000000.00   ERROR
```

max sal: 100000.00

p=8 s=2

-9.9 TO 9.9

```
Height     NUMBER(2,1)
------------
5.8
8.3
9.9
10.0   ERROR
```

Date and Time Related Data Types:

DATE:

- DATE data type is used to hold date values.
- ORACLE DATE FORMAT: DD-MON-YY.
- It can hold day, month, year, hours, minutes and seconds.
- It cannot hold fractional seconds.
- max size: 7 bytes
- it is fixed length data type.
- default time: 12:00:00 AM [midnight time]

Note:

DATE also contains time value. But, by default it will not display time.

Examples:

```
Date_Of_Joing   DATE
----------------------------
25-DEC-23
```

oredered_date   DATE

transaction_date DATE

TIMESTAMP:

- introduced in ORACLE 9i.
- It is used to hold date and time values.

- TIMESTAMP = DATE + FRACTIONAL SECONDS

- It is extension of DATE type.
- It holds day, month, year, hours, minutes, seconds and fractional seconds.
- It is fixed length data type.
- size: 11 Bytes.
- default format: DD-MON-YY HH.MI.SS.FF AM
- By default it displays time.
- default time: 12:00:00 AM

Example:
login_date_time TIMESTAMP

Differences b/w DATE and TIMESTAMP:

DATE	TIMESTAMP
it us used to hold date values	it is used to hold date and time
example: ordered_date DATE	example: txn_date_time TIMESTAMP
7 Bytes	11 Bytes
it cannot hold fractional seconds	it can hold fractional seconds
it does not display time by default.	by default it displays time

EMPLOYEE

EMPID NUMBER(4)	ENAME VARCHAR2(10)	State CHAR(2)	SAL NUMBER(8,2)	experience_summary CLOB	doj DATE	login_date_time TIMESTAMP
1234	RAJU	TG	6000.00	fdhjdsfjjdjjdsj	25-DEC-23	21-SEP-24 8:20.0.0 AM
1235	SRAVAN	AP	8000.00	fjdjdjdj		

100000.00

Wednesday, September 18, 2024 8:41 AM

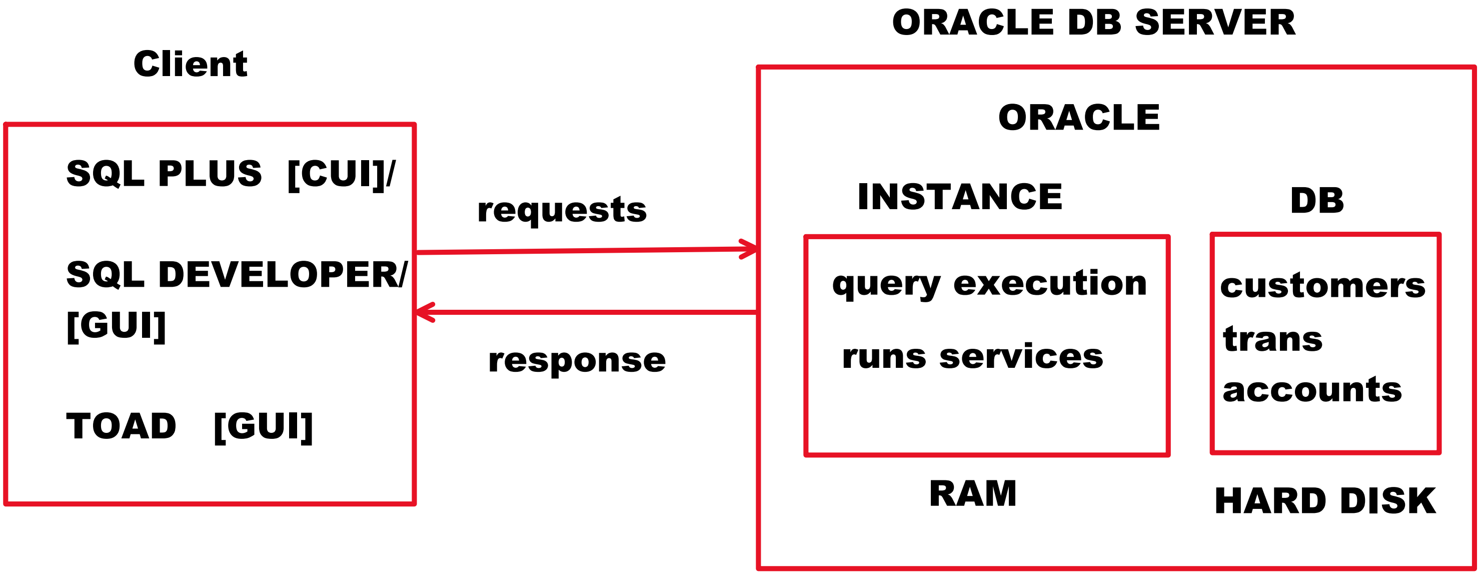
T1

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

T13

F1 LONG	F2 CLOB
VIJAY	VIJAY
SAI	SAI

Saturday, September 21, 2024 8:23 AM



Client

SQL PLUS [CUI]/

SQL DEVELOPER/
[GUI]

TOAD [GUI]

requests

response

ORACLE DB SERVER

ORACLE

INSTANCE

query execution

runs services

RAM

DB

customers
trans
accounts

HARD DISK

DB SERVER = INSTANCE + DB

queries
programs => requests

NOTE:

ORACLE:

- ORACLE is a server side software.
- The machine in which we install ORACLE software is called "ORACLE DB SERVER".
- DB SERVER contains 2 parts. They are:
  - INSTANCE [RAM]
  - DATABASE [HARD DISK]
- query processing will be done in INSTANCE [RAM]
- data will be stored permanently in DATABASE [HARD DISK]

SQL PLUS:

- It is a client side software.
- It is used to connect to DB SERVER and communicate

with DATABASE.

- **Using this software we can submit SQL queries or PL/SQL programs to DB SERVER.**

Note:
when we install ORACLE SOFTWARE, along with ORACLE, SQL PLUS software will be installed.

SQL Commands

Monday, September 23, 2024 7:49 AM

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

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

Creating User

Monday, September 23, 2024 8:03 AM

Opening SQL PLUS:

- **Press "Windows + R" [RUN]. it displays RUN dialog box.**
- **Type "SQLPLUS" command.**
- **Click on "OK". it displays SQL PLUS window.**

Creating user:

Syntax:

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

Note:
From ORACLE 12c version onwards,
there are 2 types of users. they are:

common user	**[c##raju]**
local user	**[raju]**

common user name must be prefixed with "c##"

Example:

create a common user with the name c##batch730am
with the password nareshit:

Login as DBA:

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

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

Output:
user created.

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

connect	• **it is a privilege** • **is a permission for login**

resource	•it is a privilege •is a permission to create db objects like tables, procedures, functions, packages ... etc.
unlimited tablespace	•it is a privilege •is a permission to insert the records.

Dropping User:

Syntax:

```
DROP USER <user_name> CASCADE;
```

Example:

```
DROP USER c##batch730am CASCADE;
```

to see current user name:

```
SQL> SHOW USER
```

to login from SQL command prompt:

Syntax:

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

Example:

```
SQL> conn c##batch730am/nareshit
```

to clear the screen:

Syntax:

```
CL[EAR] SCR[EEN]
```

Example:

```
SQL> CL SCR
```

Modifying password:

Syntax:

```
ALTER USER <username>
```

Syntax:

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

Example:

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

login as DBA:

username: system

```
ALTER USER c##batch730am
IDENTIFIED BY india;
```

Modifying DBA's password:

modify DBA password as tiger:

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

```
ALTER USER system
IDENTIFIED BY  tiger;
```

Creating Tables

Tuesday, September 24, 2024 7:42 AM

CREATE:

CREATE command is used to create the table.

Syntax:

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

[ ]	**Optional**
< >	**Any**

Note:

- **Till ORACLE 21c, a table can have max of 1000 columns only.**
- **In ORACLE 23ai, a table can have max of 4096 columns.**

INSERT:

- **INSERT command is used to insert the records.**

Syntax:

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

Example-1:

STUDENT

SID	SNAME	AVRG
1234	RAJU	56.78
1235	KIRAN	72.39

max avrg: 100.00

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

STUDENT

SID	SNAME	AVRG

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

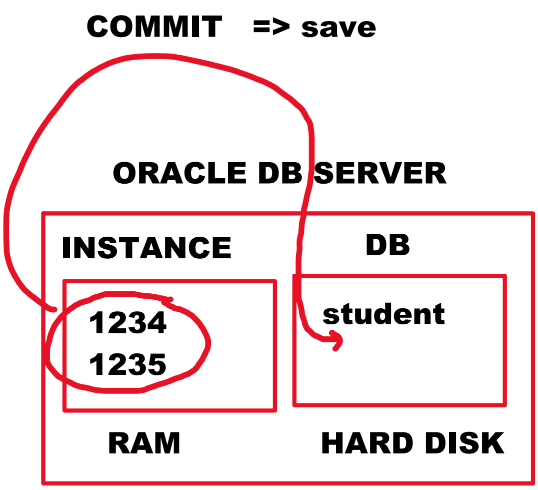
-- table will be created in DB permanently

Inserting single record:

1234	RAJU	56.78

```
INSERT INTO student
VALUES(1234, 'RAJU', 56.78);
Output:
1 row created.
```

--inserts in INSTANCE


COMMIT => save

ORACLE DB SERVER

INSTANCE

DB

1234
1235

student

RAM

HARD DISK

1235	KIRAN	72.39

```
INSERT INTO student
VALUES(1235, 'KIRAN', 72.39);
Output:
1 row created.
```

--inserts in INSTANCE

```
COMMIT;
```

--data moves from INSTANCE to DB

Inserting multiple records using parameters:

- parameter concept is used to read value at runtime.

Syntax:

```
&<text>
```

/	R[UN]	it runs recent command which is in memory

Example:

```
INSERT INTO student VALUES(&sid, '&sname', &avrg);

Output:
Enter value for sid: 2001
Enter value for sname: A
Enter value for avrg: 55.66
INSERT INTO student VALUES(&sid, '&sname', &avrg)
INSERT INTO student VALUES(2001, 'A', 55.66)
1  row created.


/
Output:
Enter value for sid: 2002
Enter value for sname: B
Enter value for avrg: 44.33
```

```
COMMIT;

SELECT * FROM student;
```

Note:
SQL is not case sensitive language.

CREATE = create = CReaTe

Inserting limited column values:

5001	ABC	

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

```
INSERT INTO student(sid, sname) VALUES(5001, 'ABC');
Output:
1 row created.
```

5002	XYZ	

```
INSERT INTO student(sid, sname) VALUES(5002, 'XYZ');
Output:
1 row created.
```

5003		45.62

```
INSERT INTO student(avrg, sid) VALUES(45.62, 5003);

COMMIT;

SELECT * FROM student;
```

To see table structure:

Syntax:

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

Example:

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

To see list of tables created by a user:

USER_TABLES:

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

```
SELECT table_name FROM user_tables;
```

SQL	ENGLISH
QUERIES	SENTENCES
CLAUSES	WORDS

Example-2:

EMPLOYEE

EMPID	ENAME	STATE	SAL	DOJ
1234	ABC	TG	15000	25-DEC-21
1235	XY	MH	12000	17-AUG-23

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

100000.00

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

Output:
Table created.

1234	ABC	TG	15000	25-DEC-21

```
INSERT INTO employee
VALUES(1234, 'ABC', 'TG', 15000, '25-DEC-2021')
```

string

```
DOJ   DATE
-------------------
25-DEC-21     date
```

implicit conversion

1235	XY	MH	12000	17-AUG-23

```
INSERT INTO employee
VALUES(1235,'XY','MH',12000, to_date('17-AUG-2023'));
```

string

DOJ DATE

to_date()

string

```
DOJ    DATE
--------------------
17-AUG-23   date
```

to_date()
Explicit Conversion

1236	AA	AP	10000	today's date

```
INSERT INTO employee
VALUES(1236, 'AA', 'AP', 10000, sysdate);

COMMIT;

select * from employee;
```

Example:

EMPLOYEE1

EMPID	ENAME	LOGIN_DATE_TIME
5001	ABC	24-SEP-24 10.30.0.0 AM
5002	XY	24-SEP-24 02.15.0.0 PM
5003	ZZZ	current sys date and time

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

5001	ABC	24-SEP-24 10.30.0.0 AM

```
INSERT INTO employee1
VALUES(5001, 'ABC', '24-SEP-2024 10.30.0.0 AM');
```

string

implicit conversion

```
LOGIN_DATE_TIME  timestamp
------------------------------
24-SEP-2024 10.30.0.0 AM        timestamp
```

24-SEP-2024 10.30.0.0 AM   timestamp

5002	XY	24-SEP-24 02.15.0.0 PM

```
INSERT INTO employee1
VALUES(5002, 'XY', to_timestamp('24-SEP-2024 2.15.0.0 PM'));
```

string

LOGIN_DATE_TIME

4-SEP-2024 2.15.0.0 PM   timestamp

to_timestamp()
explicit conversion

5003	ZZZ	current sys date and time

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

COMMIT;

SELECT * FROM employee1;
```

DATE
- implicit
- explicit
- sysdate

TIMESTAMP
- implicit
- explicit
- systimestamp

There are 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".

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

str **str**
↓ ↓
num **num**
100 + 200

Implicit conversion

Output:
300

Note:

- **DUAL is a readymade table.**
- **it is used to work with non-table data**

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

Example:

```
SELECT to_number('100')+to_number('200') FROM dual;
```

str **str**
↓ ↓ → **to_number()**
num **num**
100 + 200

explicit conversion

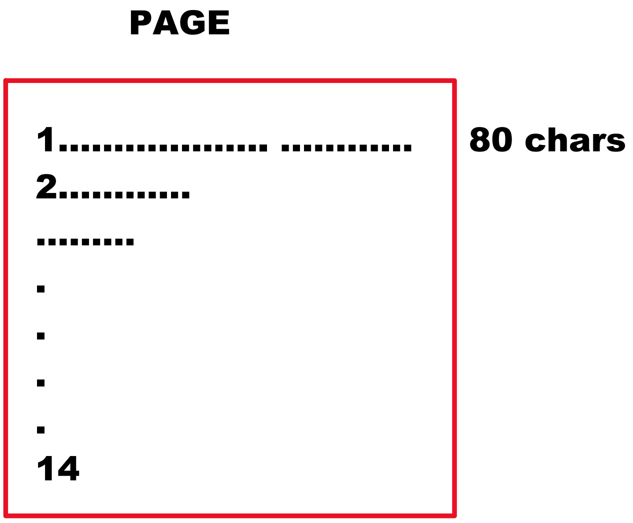
Output:
300

Note:
Don't depend on Implicit Conversion.
It degrades performance.

PAGESIZE and LINESIZE

Thursday, September 26, 2024 8:08 AM

Setting pagesize and linesize:



SQL> SHOW ALL

Output:

PAGESIZE	14
LINESIZE	80

Setting pagesize:

Syntax:

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

Example:

```
SET PAGES 200
```

Setting linesize:

Syntax:

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

Example:

```
SET LINES 200
```

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

(or)

```
SET PAGES 200 LINES 200
```

This change is applicable for entire session

COLUMN ALIAS

Friday, September 27, 2024 7:50 AM

COLUMN ALIAS:

- COLUMN ALIAS is used to change column heading in output.
- ALIAS => another name / alternative name
- To give column alias we use AS keyword
- Using AS keyword is optional.
- To give column alias in multiple words or to maintain case enclose column alias in double quotes.

Syntax:

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

Example:

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

Output:

A	B
SMITH	800
ALLEN	1600

In WINDOWS,

to search all jpg files	***.jpg**
to search all mp3 files	***.mp3**
to search jpg files which are started with 'A'	**a*.jpg**
mp3 files => names are started with S	**s*.mp3**
jpg files => in which 2nd letter is 'A'	**?a*.jpg**
jpg files => 3rd letter is a	**??a*.jpg**

*****	**0 or any num of chars**
?	**1 char**

DRL

Thursday, September 26, 2024 7:44 AM

DRL / DQL:

- **DRL => Data Retrieval Language**
- **DQL => Data Query Language**
- **It deals with data retrievals.**
- **Retrieve => opening existing data [get back]**

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

SELECT:

- **SELECT command is used to retrieve [select/fetch] the data from table.**

Syntax:

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

```
SQL
  QUERIES
       CLAUSES
```

```
ENGLISH
    SENTENCES
         WORDS
```

CLAUSE => is a part of query

NOTE:

- **Till ORACLE 21c, FROM clause is mandatory.**
- **In ORACLE 23ai, FROM clause is optional.**

Example:

```
SELECT 100+200 FROM dual;  --oracle 21c

SELECT 100+200;      --oracle 23ai
```

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

case-1:
all columns, all rows:

Display all columns and all rows of emp table:

```
SELECT * FROM emp;
```

NOTE:

*	**All Columns**

```
SELECT * FROM emp
```

above query will be rewritten by oracle as following:

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

"*" will be replaced with all column names

case-2:
all columns, specific rows:

Display the emp record whose empno is 7934:

```
SELECT * FROM emp WHERE empno=7934;
```

case-3:
specific columns, all rows:

Display all emp names and salaries:

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

case-4:
specific columns, specific rows:

Display emp name and salary of empno 7934:

```
SELECT ename, sal
FROM emp
WHERE empno=7934;
```

All Columns	SELECT *
Specific Columns	SELECT ename, sal
All rows	Don't write WHERE condn
Specific rows	write WHERE condn

Execution Process:

```
SELECT ename, sal
FROM emp
WHERE empno=7934;
```

Execution order:

FROM
WHERE
SELECT

EMP

EMPNO	ENAME	SAL	DOJ
7369	SMITH	800	17-DEC-80
7499	ALLEN	1600	20-FEB-81
7934	MILLER	1500	25-AUG-83

FROM emp:

- It selects entire emp table

EMP

EMPNO	ENAME	SAL	DOJ
7369	SMITH	800	17-DEC-80
7499	ALLEN	1600	20-FEB-81
7934	MILLER	1500	25-AUG-83

WHERE empno=7934:
WHERE is used to filter the rows.
WHERE condition will be applied on every row

EMP

EMPNO	ENAME	SAL	DOJ
7369	SMITH	800	17-DEC-80
7499	ALLEN	1600	20-FEB-81
7934	MILLER	1500	25-AUG-83

WHERE empno=7934
--

7369 = 7934	FALSE
7499 = 7934	FALSE
7934 = 7934	TRUE

EMPNO	ENAME	SAL	DOJ
7934	MILLER	1500	25-AUG-83

SELECT ename, sal:

- **SELECT selects specified columns**

ENAME	SAL
MILLER	1500

OPERATORS IN ORACLE SQL:

OPERATOR:

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

ORACLE SQL provides following Operators:

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

Arithmetic Operators:

Arithmetic operators are used to perform arithmetic operations.

ORACLE SQL provides following Arithmetic Operators:

+ - * /

Examples on Arithmetic Operators:

Calculate Annual Salary:

SELECT ename, sal, sal*12 FROM emp;

Output:

ENAME	SAL	SAL*12
SMITH	800	9600
ALLEN	1600	19200

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

Output:

ENAME	SAL	ANNUAL_SAL
SMITH	800	9600
ALLEN	1600	19200

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

Display today's date:

SELECT sysdate FROM dual;

Note:
sysdate:

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

Calculate experience of all emps:

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

sysdate-hiredate	num of days
(sysdate-hiredate)/365	num of years
trunc()	built-in function it is used to remove decimal places Example: TRUNC(123.6789) => 123

Calculate TA, HRA, TAX and GROSS salary of all emps.

10% on sal as TA
20%   HRA
2%   TAX
GROSS = sal+ta+hra-tax

ENAME	SAL	TA	HRA	TAX	GROSS

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

Relational Operators:

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

Syntax:

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

Examples:

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

Examples on Relational Operators:

Display all managers records:

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

MANAGER=manager FALSE

Output;
no rows selected.

Note:

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

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

Output:
displays managers records

Display 7521 emp record:

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

Display BLAKE record:

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

Display the emp records whose salary is 3000 or more:

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

Display the emp records whose salary is 1250 or less:

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

Display all emp records except managers:

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

```
JOB            WHERE job!='MANAGER'
========       =====================
CLERK          CLERK != MANAGER       T
MANAGER        MANAGER != MANAGER     F
ANALYST        ANALYST != MANAGER     T
```

Display 20th dept records:

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

Note:
CALENDAR order is ascending order => small to big

2022 Calendar

1-JAN-2022   => min
2-JAN-2022
3-JAN-2022
.
.
31-DEC-2022   => max

1-JAN-2023
2-JAN-2023
.
.
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';
```

```
hiredate          hiredate>'31-DEC-1981'
----------------  --------------------------------------
25-DEC-1980       25-DEC-1980 > 31-DEC-1981   F
17-FEB-1983       17-FEB-1983 > 31-DEC-1981   T
```

Display the emp records who joined before 1981:

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

hiredate<'1-JAN-1981'

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

Logical Operators:

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

ORACLE SQL provides following logical operators. They are:

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

AND:

- **It is used to perform logical AND operations**

  Syntax:

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

OR:

- **It is used to perform logical OR operations**

  Syntax:

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

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

TRUTH TABLE:

c1	condition1
c2	condition2

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

Examples on Logical Operators:

Display all managers and clerks records:

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

Output:

```
no rows selected  => All F
```

```
JOB                 job='MANAGER' AND job='CLERK'
-------             ---------------------------------------------------
ANALYST             F                                               F
CLERK               F                                               F
MANAGER             T                         F                     F
```

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

Display the emp records who are working

in deptno 10 and 30:

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

deptno	deptno=10 OR deptno=30		
------------	--------------------------------------		
20	F	F	F
10	T		T
30	F	T	T

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

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

Display the emp records who joined in 1982:

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

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

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

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

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

Display the managers who are earning more than 2500:

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

Display the managers records who joined after may 1981:

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

Display the managers whose salary is more than 2500 and who joined after april 1981:

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

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

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

Display the managers and clerks records who are earning more than 2500:

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

Online shopping App

FILTER

search for DELL and MICROSOFT

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

jul, aug, sep => 3 months transactions

```
SELECT *
FROM transactions
WHERE txn_date>='1-JUL-2024' AND txn_date<=sysdate;
```

NOT:

- it is used to perform logical NOT operations

TRUTH TABLE:

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

Display all emp records except managers:

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

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

Display all emp records except deptno 30 emps:

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

IN:

Syntax:

`<column> IN(<value_list>)`

Example:

sal IN(1250, 2450, 3000)

SAL	sal IN(1250, 2450, 3000)	
5000	5000	F
2450	2450	T
2000	2000	F
3000	3000	T

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

Examples on IN:

Display the emp records whose salaries are 1250, 2450, 3000:

```
SELECT ename, sal
FROM emp
WHERE sal=1250 OR sal=2450 OR sal=3000;
```

(or)

```
SELECT ename, sal
FROM emp
WHERE sal IN(1250, 2450, 3000);
```

Display all managers and clerks records:

ENAME	JOB	SAL

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

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

ENAME	SAL	DEPTNO

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

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

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

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

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

Display all emp records except managers and clerks:

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

```
JOB                 job NOT IN('MANAGER', 'CLERK')
---------           ------------------------------------------
MANAGER             MANAGER    F
SALESMAN            SALESMAN  T
CLERK               CLERK        F
ANALYST             ANALYST    T
```

Display the emp records who are not working in deptno 10 and 30:

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

BETWEEN AND:

Syntax:

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

Example:

```
sal BETWEEN 2450 AND 3000
```

```
SAL             sal BETWEEN 2450 AND 3000
-------         ---------------------------------------------------
5000            5000   F
2500            2500   T
3000            3000   T
1000            1000   F
```

- **BETWEEN AND operator is used to compare column value with a range values.**

- **If column value falls between range then condition is TRUE.**
- **If column value not falls between range then condition is FALSE.**

Examples on BETWEEN AND:
Display the emp records whose salary is 2450 or more and 3000 or less [sal should be b/w 2450 and 3000]:

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

(or)

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

Display the emp records who joined in 1982:

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

ename	sal	hiredate

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

Display the emp records whose empnos between 7500 to 7800:

```
SELECT * FROM emp
WHERE empno BETWEEN 7500 AND 7800;
```

Display the emp records whose salary is less than 1000 or more than 3000 [sal should not be b/w 1000 and 3000]:

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

(or)

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

LIKE:

Syntax:

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

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

%	0 or any num of chars
_	1 char

Examples on LIKE:

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

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

Display the emp records whose names are ended with S:

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

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

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

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

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

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

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

Display the emp records whose names are having A letter:

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

Display the emp records whose names are having 4 letters:

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

Display the emp records who 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 names 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 names are having %:

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

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

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

IS NULL:

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

Example:
Display the emp records who are not getting commission:

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

Output:

no rows selected [wrong result]

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

Display the emp records who are getting comm:

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

Concatenation Operator:

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

Syntax:

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

Examples:

'RAJ' || 'KUMAR'	RAJKUMAR
'RAJ' || ' ' || 'KUMAR'	RAJ KUMAR

Example:

EMPLOYEE1

EMPID	FNAME	LNAME
1001	RAVI	TEJA
1002	SAI	KRISHNA

```
SELEC empid, FNAME || ' ' || LNAME AS ename
FROM employee1;
```

EMPID	FNAME	LNAME

FNAME || ' ' || LNAME

EMPID	FNAME	LNAME
1001	RAVI	TEJA
1002	SAI	KRISHNA

```
FNAME || ' ' || LNAME
------------------------------------
RAVI TEJA
SAI KRISHNA
```

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

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

NULL

Tuesday, October 1, 2024 8:38 AM

NULL:

- **NULL means empty or blank.**

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

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

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

  Example:

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

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

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

  Display the emp records who are not getting commission:

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


  SELECT ename, sal, comm
  FROM emp
  ```

```
WHERE comm IS null;
```

Inserting Nulls:

2 ways:

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

Example:

EMPLOYEE

EMPID	ENAME	SAL

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

Direct way:

1234	RAJU	

```
INSERT INTO employee VALUES(1234, 'RAJU', NULL);
```

1235		8000

```
INSERT INTO employee VALUES(1235, NULL, 8000);
```

1236		7000

```
INSERT INTO employee VALUES(1236, '', 7000);
```

Indirect way:

1237	KIRAN	

```
INSERT INTO employee(empid,ename) VALUES(1237, 'KIRAN');
```

Tuesday, October 1, 2024 8:41 AM

STUDENT

SID	SNAME	M1 NUMBER(3)
1001	A	67
1002	B	0
1003	C	55
1004	D	

case-1:

unable to insert ABSENT
So insert NULL

EMPLOYEE

EMPID	ENAME	SAL
1001	A	12000
1002	B	15000
1003	C	

case-2:

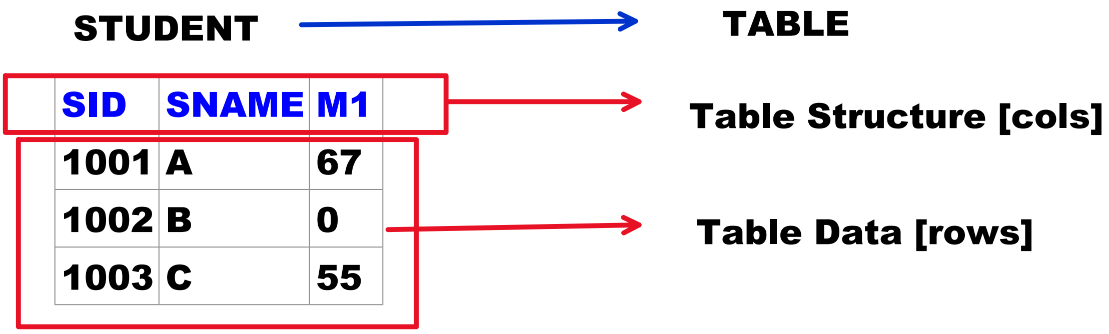
sal is unknown
so, insert NULL

UPDATE

Wednesday, October 2, 2024 8:18 AM

UPDATE:

TABLE = TABLE STRUCTURE + TABLE DATA



STUDENT → TABLE

SID	SNAME	M1
1001	A	67
1002	B	0
1003	C	55

- **UPDATE command is used to modify table data.**
- **Using UPDATE command we can modify:**
    - **single value of single record**
    - **multiple values of single record**
    - **specific group of records**
    - **all records**

Syntax:

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

SQL
QUERIES
CLAUSES

ENGLISH
SENTENCES
WORDS

CLAUSE => is a part of query

Modifying single value of single record:

Increase 2000 rupees salary to an employee whose empno is 7499:

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

COMMIT;
```

Modifying multiple values of sing record:

Set job as manager, sal as 7000 to an employee whose empno is 7369:

```
UPDATE emp
SET job='MANAGER', sal=7000
WHERE empno=7369;
```

Modifying specific group of records:

Increase 20% on salary to all managers:

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

Modifying all records:

Increase 2000 rupees salary to all emps:

```
UPDATE emp
SET sal=sal+2000;
```

Examples on UPDATE:

Transfer 10th dept emps to 20th dept:

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

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

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

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

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

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

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

In C/ Java:

```
x=50
x==50
```

=	assignment
==	comparison

In SQL:

=	assignment
=	comparison

SET empno=1234	assignment
WHERE empno=7369	comparison

Increase 20% on sal to the emps who are working in deptno

10 and 30:

```
UPDATE emp
SET sal=sal+sal*0.2
WHERE deptno IN(10,30);
```

Increase 10% on sal to the emps who are having more than 42years experience:

```
UPDATE emp
SET sal=sal+sal*0.1
WHERE TRUNC((sysdate-hiredate)/365)>42;
```

Increase 15% on salary to the emps who are getting annual salary as more than 40000:

```
UPDATE emp
SET sal=sal+sal*0.15
WHERE sal*12>40000;
```

Increase 10% on salary to the emps who joined in 1982:

```
UPDATE emp
SET sal=sal+sal*0.1
WHERE hiredate BETWEEN '1-JAN-1982' AND '31-DEC-1982';
```

DELETE

Thursday, October 3, 2024 7:53 AM

DELETE:

- **DELETE command is used to delete the records from table.**
- **Using DELETE command we can delete:**
  - **single record**
  - **specific group of records**
  - **all records**

Syntax:

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

Examples on delete:

deleting single record:

delete an emp record whose empno is 7788:

```
DELETE FROM emp
WHERE empno=7788;
```

deleting specific group of records:

delete all managers records:

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

delete the emp records who are working deptno 10 and 30:

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

delete the emp records who joined in 1982:

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

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

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

Deleting all records:

delete all emp records:

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

ALTER

Thursday, October 3, 2024 8:08 AM

ALTER:

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

Syntax:

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

Example:

STUDENT5

SID	SNAME

```
CREATE TABLE student5
(
sid NUMBER(4),
sname VARCHAR2(10)
);
```

```
DESC student5
Output:
NAME        TYPE
---------------------------------
SID          NUMBER(4)
SNAME        VARCHAR2(10)
```

Adding a column [add m1]:

```
ALTER TABLE student5 ADD m1 NUMBER(3);
```

```
DESC student5
Output:
NAME        TYPE
```

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

Adding multiple columns [Add m2, m3]:

```
ALTER TABLE student5
ADD(m2 NUMBER(3), m3 NUMBER(3));
```

```
DESC student5
Output:
NAME        TYPE
--------------------------------
SID            NUMBER(4)
SNAME          VARCHAR2(10)
M1             NUMBER(3)
M2             NUMBER(3)
M3             NUMBER(3)
```

Renaming column [rename m3 to maths]:

```
ALTER TABLE student5 RENAME COLUMN m3 TO maths;
```

```
DESC student5
Output:
NAME        TYPE
--------------------------------
SID            NUMBER(4)
SNAME          VARCHAR2(10)
M1             NUMBER(3)
M2             NUMBER(3)
MATHS          NUMBER(3)
```

Dropping a column [drop maths column]:

```
ALTER TABLE student5 DROP COLUMN maths;
(or)
ALTER TABLE student5 DROP(maths);
```

NOTE:
using DROP COLUMN we can drop 1 column only.
using DROP we can drop 1 column or multiple columns.

Dropping multiple columns [drop m1 and m2]:

```
ALTER TABLE student5 DROP(m1, m2);
```

```
DESC student5
Output:
```

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

Modifying field size [modify sname field size from 10 to 20]:

```
ALTER TABLE student5 MODIFY sname VARCHAR2(20);
```

```
DESC student5
```

Output:

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

Modifying data type [modify sid type from number to char]:

```
ALTER TABLE student5 MODIFY sid CHAR(7);
```

```
DESC student5
```

Output:

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

DCL

Friday, October 4, 2024 7:41 AM

DCL / ACL:

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

ORACLE SQL provides following DCL commands:

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

GRANT:

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

Syntax:

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

Examples:

```
USER c##batch730am
     TABLE emp
```

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

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

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

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

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

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

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

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

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

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

REVOKE:

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

Syntax:

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

Examples:

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

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

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

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

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

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

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

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

Cancel all permissions on emp table from all users:

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

Example on GRANT and REVOKE:

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

Login as DBA:

username: system

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

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


CREATE USER c##userB
```

IDENTIFIED BY userb;

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

Open 2 SQL PLUS windows.
Arrange them side by side. [Windows + Right Arrow]

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

c##userA:

```
create table t1
(
f1 number(4),
f2 varchar2(10)
);

insert into t1 values(1,'A');
insert into t1 values(2,'B');
commit;
```

c##userB:

```
select * from c##userA.t1;
Output:
error: table does not exist
```

c##userA:

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

c##userB:

```
select * from c##userA.t1;
Output:
```

F1	F2
1	A
2	B

```
INSERT INTO c##userA.t1
VALUES(3,'C');
```

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

```
select * from t1;
output:
```

F1	F2
1	A
2	B
3	C

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

UPDATE c##userA.t1
SET f2='sai'
WHERE f1=1;
Output:
error: insufficient privileges

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

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

COMMIT;
```

```
UPDATE c##userA.t1
SET f2='sai'
WHERE f1=1;
```

```
SELECT * FROM t1;
Output:
output:
```

F1	F2
1	sai
3	C

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

```
DESC t1
Output:
f1      ..
```

```
Output:
1 row updated.

COMMIT;
```

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

COMMIT;
```

```
ALTER TABLE c##userA.t1
ADD f3 DATE;
Output:
ERROR: insufficient privileges
```

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

```
f2    ..
f3    ..
```

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

```
INSERT  => error
UPDATE => error
DELETE => error

SELECT => displays data
```

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

```
SELECT * FROM c##userA.t1;
Output:
error: Table does not exist
```

user_tab_privs_made
user_tab_privs_recd

user_tab_privs_made:

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

```
DESC user_tab_privs_made
```

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

user_tab_privs_recd:

- **it is a system table / built-in table / readymade table.**
- **It maintains permissions info which are received by GRANTEE.**

```
DESC user_tab_privs_recd
```

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

DROP, FLASHBACK and PURGE

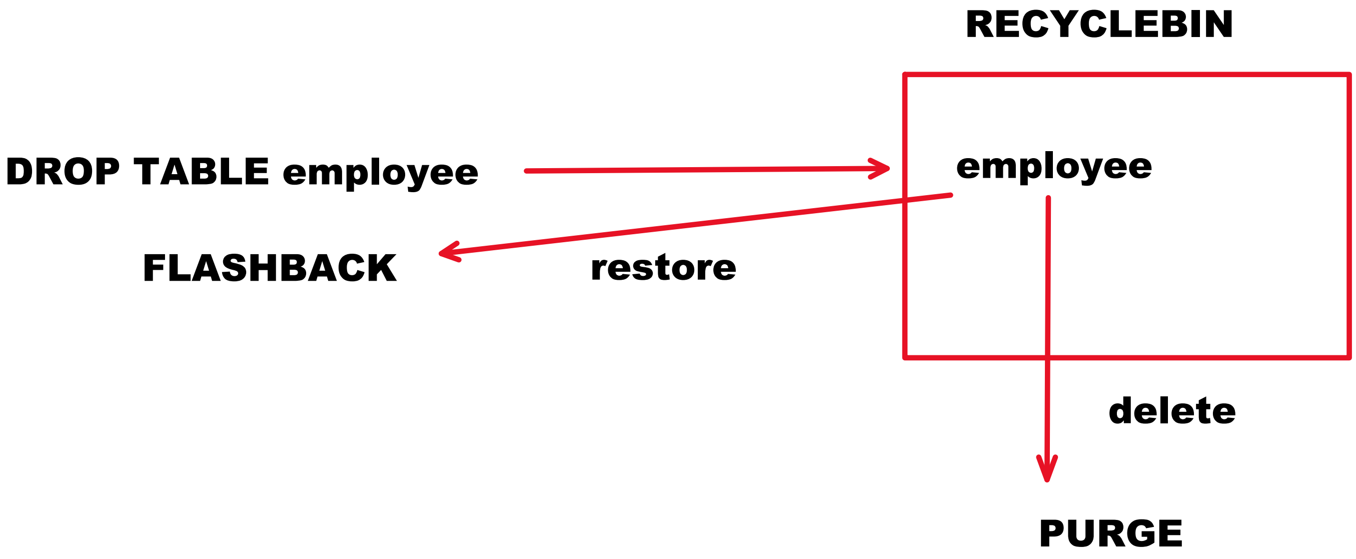
Monday, October 7, 2024 7:53 AM

DROP, FLASHBACK and PURGE:

Before ORACLE 10g:

DROP TABLE employee;
--table will be dropped permanently

From ORACLE 10g:



RECYCLEBIN

DROP TABLE employee

employee

FLASHBACK

restore

delete

PURGE

DROP:

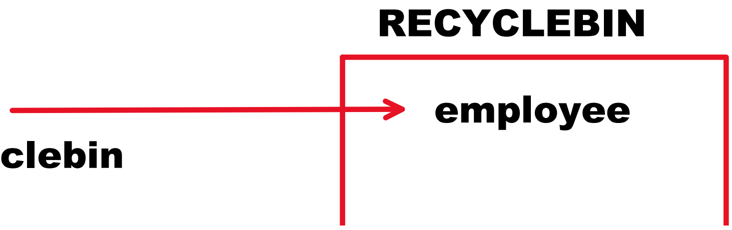
- **It is used to delete a table.**
- **when we drop the table, it will be moved to recyclebin.**

Syntax:

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

Example:

DROP TABLE employee;
--it will be moved to recyclebin

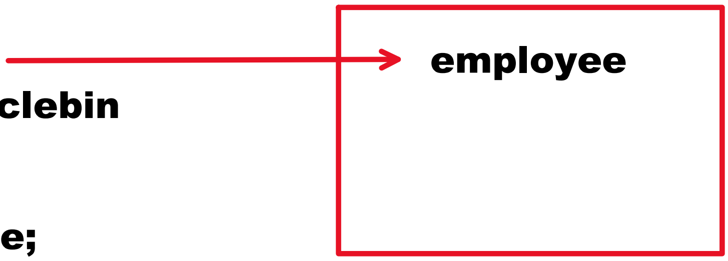


RECYCLEBIN

employee

Example:

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



employee

```
SELECT * FROM employee;
Output:
error: table does not exist
```

To see recyclebin:

```
SHOW RECYCLEBIN
Output:
TABLE_NAME
-------------------------
EMPLOYEE
```

```
DROP TABLE employee PURGE;
--employee table will be dropped permanently
```

FLASHBACK:

FLASHBACK command is used to restore the table from RECYCLEBIN.

Syntax:

```
FLASHBACK TABLE <table_name> TO BEFORE DROP;
```

Example:

```
FLASHBACK TABLE employee TO BEFORE DROP;
```

PURGE:

- **PURGE command is used to delete the table from RECYCLEBIN.**

  Syntax:

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

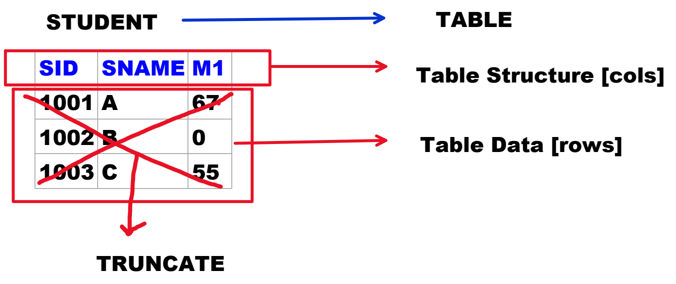
  Example:

  ```
  PURGE TABLE employee;
  ```

TRUNCATE

Monday, October 7, 2024 8:30 AM

TABLE = TABLE STRUCTURE + TABLE DATA



STUDENT → **TABLE**

SID	SNAME	M1
1001	A	67
1002	B	0
1003	C	55

Table Structure [cols]

Table Data [rows]

TRUNCATE

- **TRUNCATE command is used to delete all rows with good performance.**
- **This data will be deleted permanently.**

Syntax:

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

Example:

```
TRUNCATE TABLE employee1;
--deletes all rows
--it does not delete table structure
```

Differences b/w TRUNCATE and DROP:

TRUNCATE	• is used to delete all rows [table data]. • it does not delete table structure. • it cannot be flashed back.
DROP	• is used to delete entire table. • it deletes structure also. • it can be flashed back.

```
DELETE FROM employee;          --all rows will be deleted
TRUNCATE TABLE employee;       --all rows will be deleted
```

Differences b/w DELETE and TRUNCATE:

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

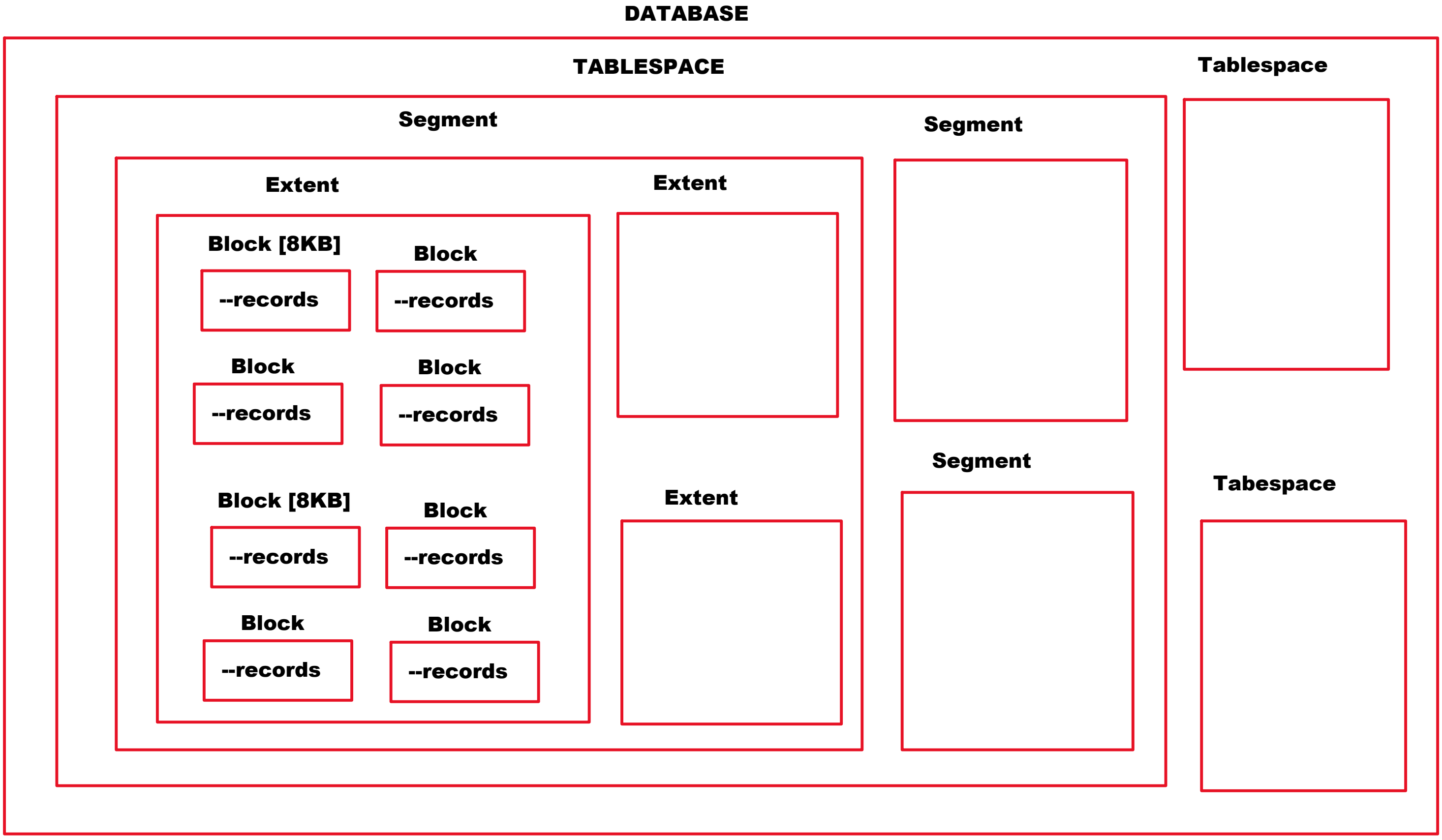
NOTE:

- **All DDL commands are auto committed.**

  DDL command = DDL command + COMMIT

  CREATE = CREATE + COMMIT
  ALTER   = ALTER + COMMIT

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



DATABASE

TABLESPACES

SEGEMNTS

EXTENTS

BLOCKS

RECORDS

RENAME

Monday, October 7, 2024 9:09 AM

RENAME:
It is used to rename the table.

Syntax:

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

Example:

```
RENAME employee TO emp1;
```

TCL

Tuesday, October 8, 2024 7:48 AM

TCL:

- **TCL => Transaction Control Language**

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

Examples:
withdraw, deposit, fund transfer, placing order

ACCOUNTS

ACNO	NAME	BALANCE
1001	A	80000
1002	B	20000

Fund transfer Transaction:

```
Begin Transaction
    sufficient funds available?      =>   select
    update from account balance =>  update
    update to account balance       =>  update
End Transaction
```

A Transaction must be successfully finished or cancelled.

Every successful transaction ends with COMMIT.
Every unsuccessful transaction ends with ROLLBACK.

ORACLE SQL provides following TCL commands:

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

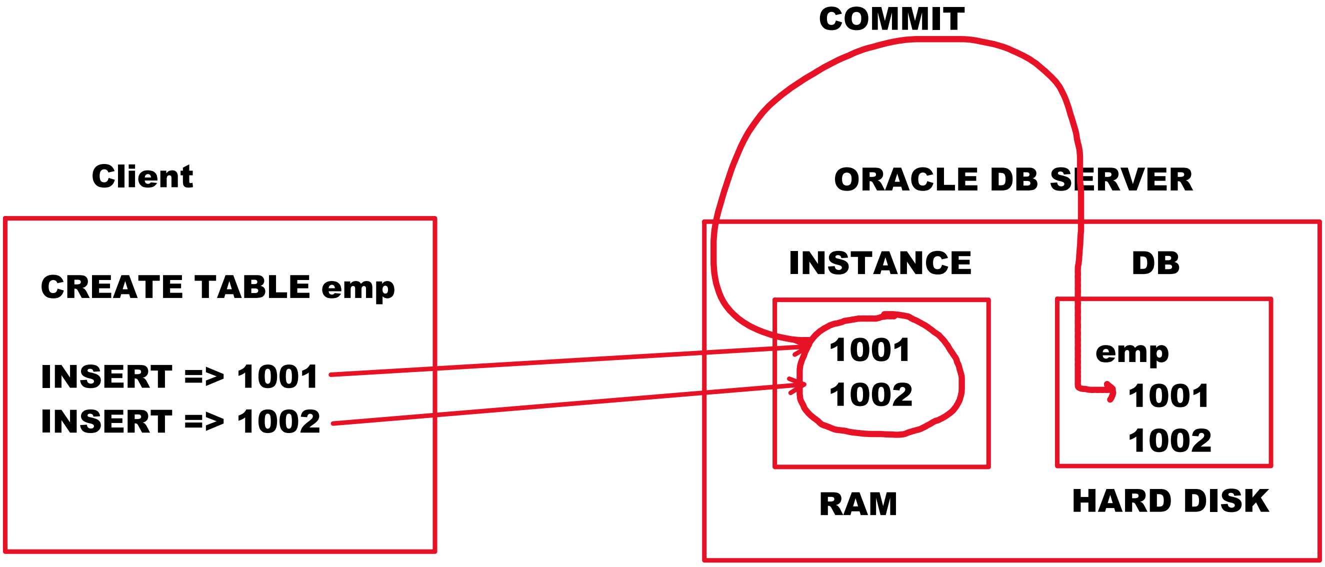
COMMIT [save]:

- **COMMIT command is used to save a transaction.**

- **When COMMIT command is executed, the data in INSTANCE will be moved to DB.**
- **It makes the changes permanent.**

Syntax:

```
COMMIT;
```



COMMIT

Client

CREATE TABLE emp

INSERT => 1001

INSERT => 1002

ORACLE DB SERVER

INSTANCE

1001

1002

RAM

DB

emp

1001

1002

HARD DISK

ROLLBACK:

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

Syntax:

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

Example on ROLLBACK and COMMIT:

Example:

```
CREATE TABLE  emp15
(
empid NUMBER(4),
ename VARCHAR2(10)
);
```

table will be created in DB

```
INSERT INTO emp15
VALUES(1001,'A');
```

it will be inserted in INSTANCE

```
INSERT INTO emp15
VALUES(1002,'B');                    it will be inserted in INSTANCE

COMMIT;                              data will be moved from INSTANCE to DB


INSERT INTO emp15
VALUES(1003,'C');

INSERT INTO emp15
VALUES(1004,'D');

ROLLBACK;                            2 actions will be cancelled
```

SAVEPOINT:

- **SAVEPOINT is used to specific point for ROLLBACK.**

Syntax:

```
SAVEPOINT <savepoint_name>;
```

Example:

```
CREATE TABLE t1(f1 INT);

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

SAVEPOINT p1;

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

SAVEPOINT p2;

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

ROLLBACK TO P2;


SELECT * FROM t1;
```

Built-In Functions

Tuesday, October 8, 2024 8:50 AM

Built-In Functions:

- To make our actions easier ORACLE SOFTWARE DEVELOPERS defined some functions and placed them in ORACLE DATABASE. These are called "Built-In Functions".

- Function => Task / Job / Activity / Action

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

- String Functions
- Conversion Functions
- Aggregate Functions [Group / Multi Row Functions]
- Number Functions
- Date Functions
- Analytic Functions [Window Functions]
- Other Functions

String Functions:

lower()
upper()
initcap()

length()
concat()

Substr()
Instr()

Ltrim()
Rtrim()
Trim()

Lpad()
Rpad()

Replace()
Translate()

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;
```

upper():

it is used to convert the string to upper case.

Syntax:

```
upper(<string>)
```

Examples:

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

initcap(): [initial capital]
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

Display all emp names and salaries.
display all 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);
```

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

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

        BLAKE=blake    FALSE
```

Output:
no rows selected.

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

Output:
displays BLAKE record

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

Substr():

- **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	J		K	U	M	A	R

Substr('RAJ KUMAR', 5)	KUMAR
Substr('RAJ KUMAR', 6)	UMAR
Substr('RAJ KUMAR', 1, 3)	RAJ
Substr('RAJ KUMAR', 6, 3)	UMA
Substr('RAJ KUMAR', 2, 4)	AJ K

2nd arg, position can be -ve.

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

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

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

1	2	3	4	5	6	7	8
S	A	I		T	E	J	A
-8	-7	-6	-5	-4	-3	-2	-1

Substr('SAI TEJA', 5, 3)	TEJ
Substr('SAI TEJA', -4, 3)	TEJ

BANK sends CREDIT CARD BILL

your password is:
your name's first 4 chars and
your credit card number's last 4 digits

SRAVAN KUMAR
1234 5678 1234 5678

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

SRAV5678

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

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

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

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

COMMIT;
```

Display the emp records whose names are started with S:

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

(or)

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

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

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

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);
```

Display the emp records whose names are started with VOWEL:

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

Display the emp records whose names are ended with VOWEL:

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

```
WHERE substr(ename,-1,1) IN('A','E','I','O','U');
```

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

Substr(ename,1,1) IN('A','E','I','O','U') **TRUE**
started with vowel

AND

substr(ename,-1,1) IN('A','E','I','O','U') **TRUE**
ended with vowel

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

Lpad() and Rpad():

- **Pad => fill**
- **L => Left**
- **R => Right**

Lpad():

- **it is used to fill the characters at left side**

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

3rd arg	default char: space

Rpad():

- **it is used to fill the characters at right side**

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

3rd arg	default char: space

Examples:

Lpad('RAJU', 10, '*')	******RAJU
Rpad('RAJU', 10, '*')	RAJU******
Lpad('SAI', 8, '$')	$$$$$SAI
Rpad('SAI', 8, '$')	SAI$$$$$
Lpad('SAI', 10, '@#')	@#@#@#@SAI

Lpad('A', 6, 'A')	AAAAAA

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

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

Example:
display message as following if acno is: 1234567890
Amount debited from acno XXXXXX7890:

```
SELECT 'Amount debited from acno ' ||
Lpad('X',6,'X') || Substr('1234567890', -4, 4)
FROM dual;
```

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

- **trim => remove**
- **L => Left**
- **R => Right**

Ltrim():

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

Syntax:

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

2nd arg	**default char is space**

Rtrim():

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

Syntax:

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

2nd arg	**default char is space**

Examples:

Ltrim('****RAJU', '*')**	**RAJU**
Rtrim('RAJU****', '*')**	**RAJU**

Ltrim('@@@##@@#@#@SAI', '@#')	**SAI**

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

Trim():

- **It is used to remove unwanted chars from left side or right side or both sides.**

Syntax:

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

Examples:

Trim(Leading '*' FROM '*****RAJU*****')	RAJU*****
Trim(Trailing '*' FROM '*****RAJU*****')	*****RAJU
Trim(Both '*' FROM '*****RAJU*****')	RAJU

Trim(' RAJU ')	RAJU

Length():

- **It is used to find string length.**
- **string length => no of chars**

Syntax:

Length(<string>)

Examples:

Length('SAI')	3
Length('RAVI TEJA')	9

Display the emp records whose names are having 4 letters:

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

(or)

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

```
ENAME           length(ename)=4
---------------  ------------------------------
ALLEN            length('ALLEN')  =>  5=4   F
WARD             length('WARD')   =>  4=4   T
```

Display the emp records whose names are having 40 letters:

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

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

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

Concat():

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

Syntax:

Concat(<string1>, <string2>)

Examples:

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

FNAME	MNAME	LNAME
RAJ	KUMAR	VARMA

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

Replace():

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

Syntax:

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

Examples:

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

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

Translate():

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

Syntax:

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

Examples:

Replace('SAI KRISHNA', 'SAI', 'XYZ')	XYZ KRISHNA
Translate('SAI KRISHNA', 'SAI', 'XYZ')	XYZ KRZXHNY
Replace('abcabcaabbccabc', 'abc', 'xyz')	xyzxyzaabbccxyz
Translate('abcabcaabbccabc', 'abc', 'xyz')	xyzxyzxxyyzzxyz

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

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

Note:

Translate() can be used to encrypt the data.

Display all emp names and salaries.

Encrypt salaries of following:

0	1	2	3	4	5	6	7	8	9
@	z	#	q	w	$	^	b	u	y

```
SELECT ename,
translate(sal, '0123456789', '@z#qw$^buy') AS sal
FROM emp;
```

Instr():

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

Syntax:

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

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

Examples:

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

Instr('RAVI TEJA', 'TEJ', 1, 1)	6
Instr('RAVI TEJA', 'RAVI', 1, 1)	1

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

Instr('RAJ KUMAR', 'UMA', 1, 1)	6
Instr('RAJ KUMAR', 'KUMAR', 1, 1)	5
Instr('RAJ KUMAR', 'SAI', 1, 1)	0

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

Instr('THIS IS HIS WISH', 'IS')	3
Instr('THIS IS HIS WISH', 'IS', 1, 3)	10
Instr('THIS IS HIS WISH', 'IS', 4, 3)	14
Instr('THIS IS HIS WISH', 'IS', 7, 3)	0

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

-16 -15 -14 -13 -12 -11 -10 -9   -8   -7   -6   -5   -4   -3   -2   -1

Instr('THIS IS HIS WISH', 'IS', -1, 1)	14
Instr('THIS IS HIS WISH', 'IS', -1, 2)	10
Instr('THIS IS HIS WISH', 'IS', -4, 2)	6

Example:

EMPLOYEE

EMPNO	ENAME
1234	RAJ KUMAR
1235	VIJAY KRISHNA

FNAME	LNAME
RAJ	KUMAR
VIJAY	KRISHNA

```
create table employee
(
empno number(4),
ename varchar2(20)
);

insert into employee values(1234, 'RAJ KUMAR');
insert into employee values(1235, 'VIJAY KRISHNA');
commit;

ALTER TABLE employee
ADD(fname VARCHAR2(10), lname  VARCHAR2(10));
```

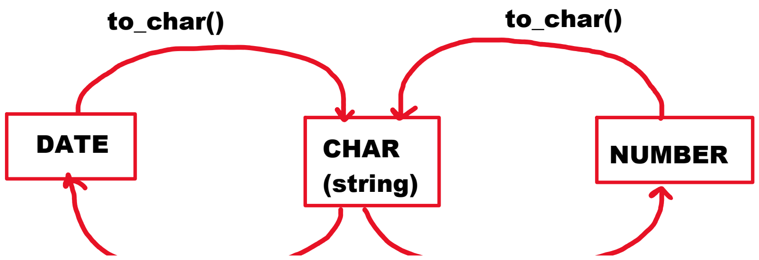
```
UPDATE employee
SET fname=substr(ename, 1, Instr(ename, ' ')-1),
lname=Substr(ename, instr(ename, ' ')+1);
```

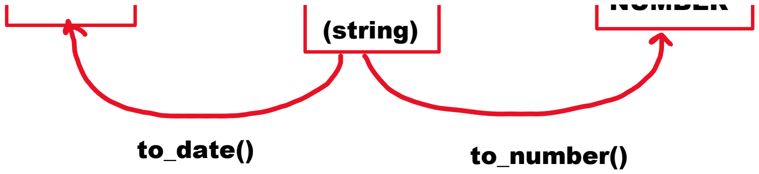
Display the emp records whose names are having AM letters:

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

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

Conversion Functions:





- to_char()
- to_date()
- to_number()

To_Char() [date to char]:

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

Syntax:

```
    To_Char(<date>, <format>)
```

FORMAT	PURPOSE	EXAMPLE sysdate: 18-OCT-24	OUTPUT
YYYY	year 4 digits	to_char(sysdate, 'YYYY')	2024
YY	year last 2 digits	to_char(sysdate, 'YY')	24

YEAR /
year
	year in words	to_char(sysdate, 'YEAR')
		to_char(sysdate, 'year')
			TWENTY TWENTY-FOUR
			twenty twenty-four

MM	month number	to_char(sysdate, 'MM')	10
MON	short month name	to_char(sysdate, 'MON')	OCT
MONTH	full month name	to_char(sysdate, 'MONTH')	OCTOBER
D	day num in week	to_char(sysdate, 'D') 1 => sun 2 => mon . . 7 => sat	6
DD	day num in month	to_char(sysdate, 'DD')	18
DDD	day num in year	to_char(sysdate, 'DDD')	31+29+31+30+31+30+31+31+30+18 = 292
DY	short weekday name	to_char(sysdate, 'DY')	FRI
DAY	full weekday name	to_char(sysdate, 'DAY')	FRIDAY
Q	quarter num jan to mar => 1 apr to jun => 2	to_char(sysdate, 'Q')	4

	jul to sep => 3 oct to dec => 4		
CC	century number	to_char(sysdate, 'CC')	21
AD / BC	AD or BC	to_char(sysdate, 'BC')	AD
HH / HH12	hours part in 12 hrs format		
HH24	hours part in 24 hrs format		
MI	minutes part		
SS	seconds part		
FF	fractional seconds	to_char(systimestamp,'FF')	
AM / PM	AM or PM		

Display current system date:

```
SELECT sysdate FROM dual;
Output:
18-OCT-24
```

Display current system date in INDIA date format:

```
SELECT to_char(sysdate, 'DD/MM/YYYY') FROM dual;
Output:
18/10/2024
```

Display current system date in US date format:

```
SELECT to_char(sysdate, 'MM/DD/YYYY') FROM dual;
Output:
10/18/2024
```

Display current system time in 12 hrs format:

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

Output:
08.49.15 AM
```

Display current system time in 24 hrs format:

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

Output:
08.49.15
```

find today's weekday:

```
SELECT to_char(sysdate, 'D') FROM dual;   --6
```

(or)

```
SELECT to_char(sysdate, 'DY') FROM dual;  --FRI
```

(or)

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

Display the emp records who joined in 1982:

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

(or)

extract year = 1982

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

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

extract year in(1980, 1982, 1984)

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

Display the emp records who joined in december mnth:

extract month = 12

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

(or)

```
SELECT ename, hiredate
FROM emp
WHERE to_char(hiredate,'MON')='DEC';
```

```
SELECT ename, hiredate
FROM emp
WHERE to_char(hiredate,'MONTH')='DECEMBER';
            DECEMBER1space = DECEMBER     FALSE
```

Output:
no rows selected

JANUARY2spaces	**9 chars**
FEBRUARY1space	**9 chars**
MARCH4spaces	**9 chars**
..	
..	
SEPTEMBER	**9 chars [max str length]**

```
SELECT ename, hiredate
FROM emp
WHERE RTRIM(to_char(hiredate,'MONTH'))='DECEMBER';
          RTRIM('DECEMBER1space')
                  DECEMBER = DECEMBER     TRUE
```

Assignment:

Display the emp records who joined in jan, may dec.

Display the emp records who joined in 4th qrtr.

Display the emp records who joined in 1st and 4th qrtrs.

Display the emp records who joined on Sunday.

Display all emp names and hiredates.
display hiredates in INDIA date format:

To_Char() [number to char]:

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

Syntax:

```
To_char(<number> [, <format> , <nls_parameters])
```

Examples:

to_char(123)	**'123'**
to_char(123.45)	**'123.45'**
to_char(5000)	**'$5000.00'**

FORMAT	**PURPOSE**
L	**currency symbol**
C	**currency name**
, (or) G	**Thousand separator**
. (or) D	**Decimal Point**
9	**Digit**

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

Display all emp names and salaries.
Apply currency symbol $, decimal point and 2 decimal places to salaries:

```
SELECT ename, tochar(sal , 'L99999.99') AS sal
FROM emp;
```

NOTE:
NLS => National Langage Support

NLS PARAMETERS	DEFAULT VALUE
NLS_CURRENCY	$
NLS_ISO_CURRENCY	AMERICA

Login as DBA:

username: system

```
SQL> SHOW PARAMETERS 'NLS'
```
Output:

NLS_CURRENCY	$
NLS_ISO_CURRENCY	AMERICA

.
.

5000 => ¥5000.00

To_Char(5000, 'L9999.99', 'NLS_CURRENCY=¥')	¥5000.00
To_Char(5000, 'C9,999.99', 'NLS_ISO_CURRENCY=JAPAN')	JPY5,000.00

To_Date():
- **It is used to convert string to date.**
- **To insert date values we need to convert string to date.**

Syntax:
```
To_Date(<string> [, <Format>])
```

Examples:

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

Example:

```
CREATE TABLE t1
(
DOJ DATE
);
```

INSERT INTO t1 VALUES(to_date('25-DEC-2023'));
string

DOJ date

25-DEC-23 date

to_date()

INSERT INTO t1 VALUES(to_date('25/12/2023', 'DD/MM/YYYY'));
string

DOJ date

25-DEC-23 date

To_Number():

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

Syntax:

To_Number(<string>, <format>)

Examples:

To_number('123')	123
To_number('123.45')	123.45
To_number('$5000.00')	ERROR
To_number('$5000.00', 'L9999.99')	5000

Find today's weekday:

SELECT To_Char(sysdate, 'DAY') FROM dual;

Find the weekday on which INDIA got INDEPENDENCE:

To_Char('15-AUG-1947', 'DAY') => error

SELECT
TO_Char(To_Date('15-AUG-1947') , 'DAY')
FROM dual;

Aggregate Functions / Group Functions:

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

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

sum(f1)	60
avg(f1)	20
max(f1)	30
min(f1)	10
count(f1)	3

sum():

it is used to find sum of a set of values.

Syntax:
sum(<column>)

avg():
it is used to find average of a set of values.

Syntax:
avg(<column>)

max():
it is used to find maximum value in a set of values.

Syntax:
max(<column>)

min():
it is used to find maximum value in a set of values.

Syntax:
min(<column>)

count():

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

Syntax:
count(* / <column>)

Examples:

Fins sum of salaries of all emps:

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

Find sum of salaries of all managers:

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

Find sum of salaries of deptno 10 and 20:

```
SELECT sum(sal) FROM emp
WHERE deptno IN(10,20);
```

Find avg sal of all emps:

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

Find avg sal of all managers:

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

Find max sal:

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

Find min sal:

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

Find max sal and min sal in all managers:

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

Find no of emps:

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

Find how many emps are getting commission:

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

Find no of clerks:

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

Find no of emps in deptno 20:

```
SELECT count(*) FROM emp
WHERE deptno=20;
```

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

```
SELECT count(*) FROM emp;     --17
SELECT count(5) FROM emp;     --17
SELECT count(20) FROM emp;   --17
```

count(*)	• it counts no of records. • it is slower.
count(5)	• it counts no of 5s. • it is faster.

Number Functions:

sqrt()
power()
sign()
abs()

Mod()

ceil()
Floor()

Trunc()
Round()

sqrt():

- **it is used to find square root value**

Syntax:

sqrt(<number>)

Examples:

sqrt(100)	10
sqrt(81)	9

power():

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

Syntax:

power(<number>, <power>)

Examples:

power(2,3)	8
power(7,2)	49

sign():

- **it is used to check sign of the number.**
- **if num is +ve, it returns 1.**
- **if num is -ve, it returns -1.**
- **if num is 0, it returns 0.**

Syntax:

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

abs():

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

Syntax:

abs(<number>)

Examples:

abs(25)	25
abs(-25)	25

Mod():

It is used to get remainder value.

Syntax:

Mod(<number>, <divisor>)

Examples:

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

Ceil():

- **it is used to get round up value**

Syntax:
Ceil(<number>)

Floor():

- **It is used to get round down value.**

Syntax:
Floor(<number>)

Examples:
456 => 456.789 => 457

Ceil(456.789)	**457**
Floor(456.789)	**456**

TRUNC() and ROUND():

TRUNC():

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

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

Examples:

TRUNC(123.45678)	**123**
TRUNC(123.45678, 1)	**123.4**
TRUNC(567.89564, 3)	**567.895**

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

-1	**rounds in 10s** **10, 20, 30, ...**
-2	**rounds in 100s**
-3	**rounds in 1000s**

TRUNC(567.789, -1)	**560 and 570** **560**
TRUNC(567.789, -2)	**500 and 600** **500**
TRUNC(3456.678, -3)	**3000 and 4000** **3000**

Round():

- **it considers avrg.**
- **if value is avrg or above avrg, it gives upper value.**

otherwise, it gives lower value.

Syntax:

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

Examples:

TRUNC(123.6789)	**123 and 124** **123**
ROUND(123.6789)	**123 and 124** **avrg of 123 and 124: 123.5** **124**
TRUNC(123.4789)	**123 and 124** **123**
ROUND(123.4789)	**123 and 124** **avrg of 123 and 124: 123.5** **123**

2nd arg can be -ve.

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

TRUNC(153.6789, -1)	**150 and 160** **150**
ROUND(153.6789, -1)	**150 and 160** **avrg of 150 and 160 => 155** **150**
TRUNC(158.6789, -1)	**150 and 160** **150**
ROUND(158.6789, -1)	**150 and 160** **avrg of 150 and 160 => 155** **160**

TRUNC(4567.789, -3)	**4000 and 5000** **4000**
ROUND(4567.789, -3)	**4000 and 5000** **avrg of 4000 and 5000: 4500** **5000**

Date Functions:

sysdate
systimestamp

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

sysdate:

- it is used to get current system date.

systimestamp:

- it is used to current system date and time.

Display today's date:

```
SELECT sysdate FROM dual;
```

Display current system time from sysdate in 12hrs format:

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

Display current system time from sysdate in 24hrs format:

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

Display current system date and time:

```
SELECT systimestamp FROM dual;
```

Extract only time from systimestamp:

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

Extract only date from systimestamp:

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

NOTE:
to remove time from date and time we can use TRUNC()

Add_Months():

- It is used to add months to specific date.
- Using it, we can also subtract months from specific date.

Syntax:

```
Add_Months(<date>, <no_of_months>)
```

Add 2 days to sysdate: sysdate: 22-OCT-24

```
SELECT sysdate+2 FROM dual;                 24-OCT-24
```

Add 2 months to sysdate:

```
SELECT add_months(sysdate, 2)               22-DEC-24
FROM dual;
```

Add 2 years to sysdate:

```
SELECT add_months(sysdate, 2*12)            22-OCT-26
```

Add 2 years to sysdate:

```
SELECT add_months(sysdate, 2*12)          22-OCT-26
FROM dual;
```

subtract 2 days from sysdate:

```
SELECT sysdate-2 FROM dual;              20-OCT-24
```

subtract 2 months from sysdate:

```
SELECT add_months(sysdate,-2)             22-AUG-24
FROM dual;
```

subtract 2 years from sysdate:

```
SELECT add_months(sysdate,-2*12)          22-OCT-22
FROM dual;
```

Examples:

ORDERS

ORDERID	PID	CID	ORDERED_DATE	DELIVERY_DATE
123456	..	..	sysdate	sysdate+5

PRODUCTS

PID	PNAME	MAN_DATE	EXP_DATE
1234	X	sysdate	Add_Months(sysdate, 3)

CMs_List

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

```
INSERT INTO emp(empno,ename,hiredate)
VALUES(6001, 'AA', sysdate);

INSERT INTO emp(empno,ename,hiredate)
VALUES(6002, 'B', sysdate-1);

INSERT INTO emp(empno,ename,hiredate)
VALUES(6003, 'C', Add_Months(sysdate,-1));

INSERT INTO emp(empno,ename,hiredate)
VALUES(6004, 'D', Add_Months(sysdate,-12));

COMMIT;
```

Display the emp records who joined today:

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

Output:

```
no rows selected
```

```
WHERE hiredate                = sysdate
           22-OCT-24 8.09 AM = 22-OCT-24 8.12 AM    FALSE
```

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

```
WHERE TRUNC(hiredate)      = TRUNC(sysdate)
TRUNC(22-OCT-24 8.09 AM)  = TRUNC(22-OCT-24 8.12 AM)
           22-OCT-24             = 22-OCT-24          TRUE
```

Display the emp records who joined yesterday:

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

Display the emp records who joined 1 month ago:

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

Display the emp records who joined 1 year ago:

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

Assignment:

GOLDRATES

DATEID	PRICE
1-JAN-2020	50000
2-JAN-2020	50800
..	
..	
22-OCT-24	80200

find today's gold rate

find yesterday's gold rate

find 1 month ago gold rate

find 1 year ago gold rate

SALES

DATEID	AMOUNT
1-Jan-2020	85000
2-JAN-2020	100000
..	..
..	
22-OCT-2024	..

find today's sales

find yesterday's sales

find 1 month ago sales

find 1 year ago sales

Months_Between():

- **it is used to find difference between 2 dates.**
- **it returns no of months.**

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

Example:

Months_Between(sysdate, '22-OCT-23')	12 [months]
Months_Between(sysdate, '22-OCT-23')/12	1 [year]

Display all emp records along with experience:

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

```
TRUNC((sysdate-hiredate)/365)
(or)
TRUNC(months_between(sysdate, hiredate)/12)
```

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

ENAME	HIREDATE	YEARS	MONTHS

15 months	TRUNC(15/12) 1 year	mod(15, 12) 3 months
30 months	TRUNC(30/12) 2 years	mod(30, 12) 6 months

ENAME	HIREDATE	YEARS	MONTHS

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

Last_day():

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

Syntax:
Last_Day(<date>)

Examples:

Last_day(sysdate)	31-OCT-24
Last_day('22-FEB-2024')	29-FEB-24
Last_day('22-FEB-2023')	28-FEB-23

Find next month first date:

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

Find current month first date:

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

Last_day(22-SEP-24)
30-SEP-24 + 1
1-OCT-24

Next_day():

- **It is find next date based on week day.**
- **to find next Friday date it is useful.**

Syntax:

```
  Next_day(<date>, <weekday>)
```

1	sun	sunday

Examples:

find next Friday date:

```
SELECT Next_day(sysdate, 'fri')
FROM dual;
```

Find next month first Sunday date:

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

Find current month last Sunday date:

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

31-OCT-24 -7
24-OCT-24

Analytic Functions / Window Functions:

Rank()
Dense_Rank()
Row_Number()

ORDER BY marks DESC

```
MARKS
-----------
678
890
950
730
950
730
```

MARKS	RANK	DENSE_RANK
950	1	1
950	1	1
890	3	2
890	3	2
890	3	2
730	6	3

730
950
730
890
500
890
400

890	3	2
890	3	2
730	6	3
730	6	3
678	8	4
500	9	5
400	10	6

RANK():

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

Syntax:

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

DENSE_RANK():

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

Syntax:

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

Examples on RANK() and DENSE_RANK():

Display all emp records.
Apply ranks to records according to sal descending order:

ENAME	SAL	RANK

```
SELECT ename, sal,
rank() over(ORDER BY sal DESC) AS rank
FROM emp;
```

(or)

```
SELECT ename, sal,
dense_rank() over(ORDER BY sal DESC) AS rank
FROM emp;
```

Display all emp records.
Apply ranks according to seniority.

ENAME	HIREDATE	RANK

```
SELECT ename, hiredate,
dense_rank() over(ORDER BY hiredate ASC) AS rank
FROM emp;
```

Display all emp records.
apply ranks to records according to salary descending order.
If salary is same don't give same rank.
If salary is same apply rank according to seniority.

ORDER BY sal DESC, hiredate ASC

ENAME	SAL	HIREDATE
A	12000	25-DEC-1983
B	10000	17-FEB-1980
C	10000	25-JUN-1981

DENSE_RANK
1
2
3

```
SELECT ename, sal, hiredate,
dense_rank() over(ORDER By sal DESC, hiredate ASC) AS rank
FROM emp;
```

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

PARTITION BY deptno ORDER BY sal DESC

deptno	sal
10	15000
10	12000
10	17000
20	20000
20	8000
20	19000

10	15000
10	12000
10	17000

20	20000
20	8000
20	19000

		rank
10	17000	1
10	15000	2
10	12000	3

20	20000	1
20	19000	2
20	8000	3

```
break on deptno skip 1 duplicates

SELECT ename, deptno, sal,
dense_rank() over(PARTITION BY deptno ORDER BY sal DESC) AS rank
FROM emp;

clear breaks
```

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

PARTITION BY job ORDER BY sal DESC

JOB	SAL
MANAGER	20000
MANAGER	25000
MANAGER	22000
CLERK	15000
CLERK	10000

MANAGER	20000
MANAGER	25000
MANAGER	22000

CLERK	15000

MANAGER	25000	1
MANAGER	22000	2
MANAGER	20000	3

CLERK	15000	1

CLERK	15000
CLERK	10000
CLERK	12000

CLERK	15000
CLERK	10000
CLERK	12000

CLERK	15000	1
CLERK	12000	2
CLERK	10000	3

ENAME	JOB	SAL	RANK

```
break on job skip 1

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

clear breaks
```

PARTITION BY	it is used to group the records according to particular column	PARTITION BY deptno
ORDER BY	it is used to arrange the records in ASC or DESC order	ORDER BY sal DESC

Row_Number():

- **it is used to apply row numbers to records.**

Syntax:

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

Examples:

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

SNO	EMPNO	ENAME	SAL

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

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

SNO	EMPNO	ENAME	SAL

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

Display all emp records.

Apply row numbers to them with in dept according to sal desc:

SNO	ENAME	DEPTNO	SAL

```
break on deptno skip 1

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

Other Functions:

NVL()
NVL2()

USER
UID

DECODE()

NVL():

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

Syntax:
NVL(<arg1>, <arg2>)

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

Examples:

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

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

SAL	COMM
800	
1600	300

```
SAL+COMM
800+null      = null
1600+300      = 1900
```

```
SAL+NVL(comm, 0)
-------------------------------
800+NVL(null, 0)  => 800+0 = 800
1600+NVL(300, 0) => 1600+300 = 1900
```

ENAME	SAL	COMM	TOTAL_SAL

```
SELECT ename, sal, comm, sal+NVL(comm,0) AS total_sal
FROM emp;
```

Display all emp details along with commissions. If comm is null display it as N/A:

ENAME	SAL	COMM
SMITH	800	N/A
ALLEN	1600	300

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

Assignment:

STUDENT

SID	SNAME	SUB1
1001	A	67
1002	B	0
1003	C	50
1004	D	

replace nulls with AB

```
NVL(to_char(sub1), 'AB')
```

NVL2():

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

Syntax:

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

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

Examples:

NVL2(10,20,30)	20
NVL2(null, 20, 30)	30

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

NVL()	•it replaces nulls only. •it can take 2 arguments.
NVL2()	•it replaces nulls and not nulls. •it can take 3 arguments.

Modify comm values as following:
if emp is getting comm then increase 1000 rupees comm.
if emp is not getting comm then set comm as 900:

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

USER:
It returns current user name

UID:
It returns current user id

```
SELECT UID, USER FROM dual;
```

DECODE():

- **It is used to implement "if" control structure in SQL.**
- **It checks equality condition only.**

Syntax:

```
DECODE(<column>,
            <value1>, <return_value1>,
           <value2>, <return_value2>,
           .
           .
           <else_value>)
```

Example:
Display all emp records along with job titles.
If job is PRESIDENT display it as BIG BOSS

MANAGER	**BOSS**
Others	**SUBORDINATE**

ENAME	JOB	SAL

```
SELECT ename,
decode(job,
          'PRESIDENT', 'BIG BOSS',
          'MANAGER', 'BOSS',
          'SUBORDINATE') AS job,
sal
FROM emp;
```

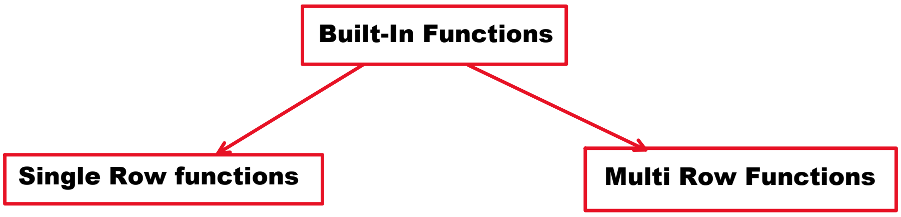
String Functions	**lower() upper() initcap()**
	length() Lpad() Rpad()
	Ltrim() Rtrim() Trim()
	Replace() Translate()
	Substr() Instr()

Conversion	**to_char() to_date() to_number()**
Aggregate / group	**sum() max() min() count() avg()**

Number	**ceil() floor()**
	trunc() round() mod()

Date	**sysdate systimestamp** **add_months() months_between()**

	last_day() next_day()
Analytic	rank() dense_rank() row_number()
Other	NVL() NVL2() USER DECODE()


Built-In Functions

Single Row functions

Multi Row Functions

string functions
number functions
conversion functions
.
.

aggregate functions

Single Row Function:
If one function call is applied on one row

Multi Row Function:
If one function call is applied on multiple rows

```
ENAME          lower(ename)
--------------  ---------------------
SMITH          lower('SMITH')   => smith
ALLEN          lower('ALLEN')   => allen
WARD           lower('WARD')    =>  ward
```

```
SAL               MAX(SAL) => 15000
----------
10000
15000             3 rows
12000
```

EDITING QUERY

Wednesday, October 23, 2024 8:14 AM

SYNTAX:

ED[IT]

- **type ED. displays editor.**
- **Edit the query.**
- **Don't write ; at end of query.**
- **save it.**
- **close the editor.**
- **type /.**

CLAUSES

Thursday, October 24, 2024 8:52 AM

SQL	ENGLISH
QUERIES	SENTENCES
CLAUSES	WORDS

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

SELECT command clauses are:

- FROM
- WHERE
- SELECT
- ORDER BY
- GROUP BY
- HAVING
- OFFSET [oracle 12c]
- FETCH [oracle 12c]
- DISTINCT

Syntax of SELECT command:

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

SELECT	it is used to specify column names. Example: SELECT ename, sal

FROM	it is used to specify table names.
	Examples:
	FROM emp
	FROM emp, dept

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

	WHERE job='MANAGER'
	WHERE deptno IN(10,30)

ORDER BY:

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

Syntax:

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

Examples on ORDER BY:

Display all records. arrange emp names in alphabetical order:

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

(or)

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

(or)

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

1	ename
2	sal

Display all emp records.
Arrange them in Descending order according to salary:

ENAME	SAL [DESC]

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

(or)

```
SELECT ename, sal
```

```
FROM emp
ORDER BY 2 DESC;
```

Display all emp records.
Arrange them according to seniority:

ENAME	HIREDATE

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

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

ENAME	DEPTNO	SAL

```
BREAK ON deptno SKIP 1 DUPLICATES
```

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

Arranging records in order according to multiple columns:

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

ENAME	DEPTNO	SAL

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

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

ENAME	DEPTNO	SAL	HIREDATE

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

NOTE:

- **In Ascending order nulls will be displayed last.**
- **In Descending order nulls be displayed first.**

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

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

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

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

GROUP BY:

```
NARESH IT =>                    oracle 730am batch
Boss => Admin                   M ?
                                F  ?
```

```
student         GROUP BY gender
GENDER
-------------
                        M
M                       M           count(*)   => 3
F                       M
F
M                       F
M                       F           count(*)   => 3
F                       F
```

```
oracle 730am batch

OFFLINE   ?
ONLINE    ?
```

```
STUDENT
   learing_mode              GROUP BY learning_mode
   -----------------------
   ONLINE
   OFFLINE                      ONLINE
   OFFLINE                      ONLINE          count(*)
   ONLINE                       ONLINE
   ONLINE
                                OFFLINE
                                OFFLINE         count(*)
```

TG	?
MH	?
AP	?

GROUP BY state

TG
TG

sum(fee)
count(*)

MH
MH

AP
AP

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

Example:

detailed data → GROUP BY deptno → summarized data

EMP

EMPNO	ENAME	DEPTNO	SAL
1001	A	10	20000
1002	B	20	10000
1003	C	20	25000
1004	D	10	30000

10

10	20000
10	30000

sum(sal)

20

20	10000
20	25000

sum(sal)

deptno	sum_of_sal
10	50000
20	35000

Examples on GROUP BY:

Find dept wise sum of salaries:

DEPTNO	SUM_OF_SAL
10	?
20	?

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

Find dept wise no of emps:

DEPTNO	NO_OF_EMPS
10	?
20	?

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

Find dept wise max sal and min sal:

DEPTNO	MAX_SAL	MIN_SAL
10	?	?
20	?	?

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

Find job wise sum of salaries:

JOB	SUM_OF_SAL
MANAGER	?
CLERK	?

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

Assignment:
Find job wise max sal and min sal

JOB	MAX_SAL	MIN_SAL
MANAGER	?	?
CLERK	?	?

GROUP BY job
max(sal), min(Sal)

Find Job wise no of emps:

JOB	NO_OF_EMPS
CLERK	?
MANAGER	?

GROUP BY job
count(*)

Find Year wise no of emps joined in organization:

YEAR	NO_OF_EMPS
1980	?
1981	?

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

Assignment:

Find quarter wise no of emps joined in org:

QUARTER	NO_OF_EMPS
1	?
2	?
3	?
4	?

GROUP BY to_char(hiredate,'Q')

Grouping Records according to multiple columns:

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

DEPTNO	JOB	NO_OF_EMPS
20	ANALYST	?
20	CLERK	?
30	CLERK	?
30	SALESMAN	?

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

Rollup() and Cube():

Rollup():

- **It is used to calculate sub totals and grand total.**
- **It calculates subtotals according to first column.**
- **It is called from GROUP BY.**
- **we pass grouping columns as arguments.**

Syntax:

GROUP BY Rollup(<grouping column list>)

Example:

GROUP BY Rollup(deptno, job)

Cube():

- **It is used to calculate sub totals and grand total.**
- **It calculates subtotals according to all columns.**
- **It is called from GROUP BY.**
- **we pass grouping columns as arguments.**

Syntax:

GROUP BY Cube(<grouping column list>)

Example:

GROUP BY Cube(deptno, job)

Example:

Find dept wise, with in dept job wise no of emps.
Calculate sub totals and grand total.
Calculate sub totals according to deptno:
[Rollup()]

DEPTNO	JOB	NO_OF_EMPS
20	ANALYST	?
20	CLERK	?
	10th dept sub total	?
30	CLERK	?
30	SALESMAN	?
	20th dept sub total	?
	GRAND TOTAL	?

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

Find dept wise, with in dept job wise no of emps.
Calculate sub totals and grand total.
Calculate sub totals according to deptno:
[Rollup()]

DEPTNO	JOB	NO_OF_EMPS

20	ANALYST	?
20	CLERK	?
	10th dept sub total	?
30	CLERK	?
30	SALESMAN	?
	20th dept sub total	?
	ANALYST sub total	?
	CLERK sub total	?
	SALESMAN sub total	?
	GRAND TOTAL	?

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

Example:

Find year wise, with in year quarter wise no of emps.

YEAR	QUARTER	NO_OF_EMPS
1981	1	?
	2	?
	3	?
	4	?
1982	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 1;
```

Find year wise, with in year quarter wise no of emps.
calculate sub totals according to year.

YEAR	QUARTER	NO_OF_EMPS
1981	1	?
	2	?
	3	?

	4	?
	1981 sub total	?
1982	1	?
	2	?
	3	?
	4	?
	1982 sub total	?
	GRAND TOTAL	?

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

Find year wise, with in year quarter wise no of emps.
calculate sub totals according to year and quarter:

YEAR	QUARTER	NO_OF_EMPS
1981	1	?
	2	?
	3	?
	4	?
	1981 sub total	?
1982	1	?
	2	?
	3	?
	4	?
	1982 sub total	?
	1st qrtr sub total	?
	2nd qrtr sub total	?
	3rd qrtr sub total	?
	4th qrtr sub total	?
	GRAND TOTAL	?

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

Assignment:

SALES

DATEID	AMOUNT
1-JAN-2022	90000
2-JAN-2022	100000
..	..
..	
26-OCT-2024	98000

Find year wise quarter wise sales.

2022	1	?
	2	?
	sub total	?
2023	1	?
	2	?
	subtotal	?
	1 qrtr sub total	?
	2 qrtr sub total	?
	GRAND TOTAL	?

PERSON

PID	PNAME	STATE	GENDER	AADHAR

state wise no of people => GROUP BY state
gender wise no of people => GROUP BY gender

state wise, with in state gender wise no of people

TG	M	?
	F	?
AP	M	?
	F	?

GROUP BY state, gender

TG	M	?
	F	?
	TG population	?
AP	M	?
	F	?
	AP population	?
	INDIA population	?

GROUP BY Rollup(state, gender)

TG	M	?
	F	?
	TG population	?
AP	M	?
	F	?
	AP population	?
	M sub total	?

GROUP BY Cube(state, gender)

	F sub total	?
	INDIA population	?

HAVING:

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

Syntax:

```
HAVING <group_condition>
```

Examples on HAVING:

Display the depts which are spending more than 25000 rupees amount on their emps:

```
SELECT deptno, sum(sal)
FROM emp
GROUP BY deptno
HAVING sum(Sal)>25000;
```

Display the depts which are having more than 5 emps:

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

Differences between WHERE and HAVING:

WHERE	HAVING
• WHERE condition will be applied on rows.	• HAVING condition will be applied on groups.
• it filters the rows.	• it filters the groups.
• Aggregate functions cannot be used here.	• Aggregate functions can be used here.
• It gets executed before GROUP BY	• It gets executed after GROUP BY

Execution order of clauses [ORACLE 21c]:

FROM
WHERE
GROUP BY
HAVING
SELECT
DISTINCT
ORDER BY
OFFSET
FETCH

DISTINCT:

- **It is used to eliminate duplicate records.**

Syntax:

```
SELECT ALL/DISTINCT <column_list>

ORDER BY   .. ASC / DESC
```

Display the job titles offered by company:

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

Display the deptnos which are having emps:

```
SELECT DISTINCT deptno FROM emp
ORDER BY 1;
```

OFFSET:

- **introduced in oracle 12c version.**
- **It is used to specify number of rows to be skipped.**

Syntax:

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

FETCH:

- **introduced in oracle 12c version.**
- **It is used to specify number of rows to be fetched.**

Syntax:

```
FETCH FIRST/NEXT <number> ROW/ROWS ONLY
```

Examples on OFFSET and FETCH:

Display all emp records except first 5 rows:

```
SELECT * FROM emp
OFFSET 5 ROWS;
```

Display first 5 rows from emp table:

```
SELECT * FROM emp
FETCH FIRST 5 ROWS ONLY;
```

Display 6th row to 10th row:

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

Find 2nd max sal:

SAL	ORDER BY sal DESC	DISTINCT sal	

4000	5000	5000	OFFSET 1 ROW
5000	5000	4000	FETCH NEXT 1 ROW ONLY
3000	4000	3500	
3500	4000	3000	4000
5000	4000	2000	
4000	3500		
2000	3000		
4000	2000		

```
SELECT DISTINCT sal
FROM emp
ORDER BY sal DESC
```

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

Find 3rd max sal:

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

Find top 3 salaries:

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

Display top 3 seniors records:

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

FROM	selects entire table FROM emp
WHERE	filters the rows WHERE sal>3000

GROUP BY	to group the records according to specific col(s)
	GROUP BY deptno
	GROUP BY deptno, job

HAVING	to write conditions on groups HAVING sum(sal)>10000
SELECT	to specify column names SELECT ename, sal
DISTINCT	to eliminate duplicates SELECT DISTINCT job

ORDER BY	**to arrange records in asc or desc**
	default order is ASC
	ORDER BY sal DESC
	ORDER BY ename ASC

OFFSET	**to skip the rows** **OFFSET 5 ROWS**
FETCH	**to fetch the rows** **FETCH FIRST 5 ROWS ONLY**

Can we use column alias in GROUP BY?
Till ORACLE 21C, we cannot use column alias in GROUP BY.
Because of execution order.
GROUP BY gets executed before SELECT.

FROM
WHERE
GROUP BY
HAVING
SELECT
DISTINCT
ORDER BY
OFFSET
FETCH

Can we use column alias in ORDER BY?
Yes.
Because of execution order.
ORDER BY gets executed after SELECT.

NOTE:
in ORACLE 23ai, a new feature added. i.e:
We can use column alias in GROUP BY and HAVING.

find year wise no of emps joined in organization:

YEAR	NO_OF_EMPS
1980	?
1981	?

```
SELECT to_char(hiredate,'YYYY') AS year,
count(*) AS no_of_emps
FROM emp
GROUP BY year
ORDER BY year;
```

in oracle 21c => error

in oracle 23ai => it works
in 23ai, we can use
column alias in GROUP BY

```
SELECT to_char(hiredate,'YYYY') AS year,
count(*) AS no_of_emps
FROM emp
GROUP BY to_char(hiredate,'YYYY')
ORDER BY year;
```

NOTE:

```
SELECT ename, max(Sal) FROM emp;
```

Output:

ERROR

when we use group function,
SELECT clause accepts either GROUP BY column or GROUP FUNCTION.

```
SELECT deptno, ename, max(sal)
FROM emp
GROUP BY deptno
```

when we use GROUP BY,
SELECT clause accepts either GROUP BY column or GROUP FUNCTION.

JOINS

Monday, October 28, 2024 8:44 AM

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

COLLEGE DB

STUDENT
MARKS
FEE
STAFF
.
.

student.sid = marks.sid

STUDENT

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

MARKS

SID	Maths	Phy	Che
1001	76	58	67
1002	55	34	80

SID	SNAME	MATHS

STUDENT MARKS

- **JOIN is an Operation.**
- **In Join Operation, it combines (joins) one table row with another table row 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.**

Types of Joins:

- **Inner Join**
  - **Equi Join**
  - **Non-Equi Join**
- **Outer Join**
  - **Left Outer Join**
  - **Right Outer Join**

    - Full Outer Join
- Self-Join
- Cross Join

NOTE:
Inner Join = matched records
Outer Join = matched + unmatched records

Inner Join:

- Inner join gives matched records only.
- It has 2 sub types. They are:
    - Equi Join
    - Non-Equi Join

Equi Join:

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

Example:

e.deptno = d.deptno

EMP e

EMPNO	ENAME	SAL	DEPTNO
7369	SMITH	..	20
7499	ALLEN	..	30
7521	WARD	..	30
7788	SCOTT	..	10
1001	A	..	
1002	B	..	

DEPT d

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

ENAME	SAL	DNAME	LOC

EMP DEPT

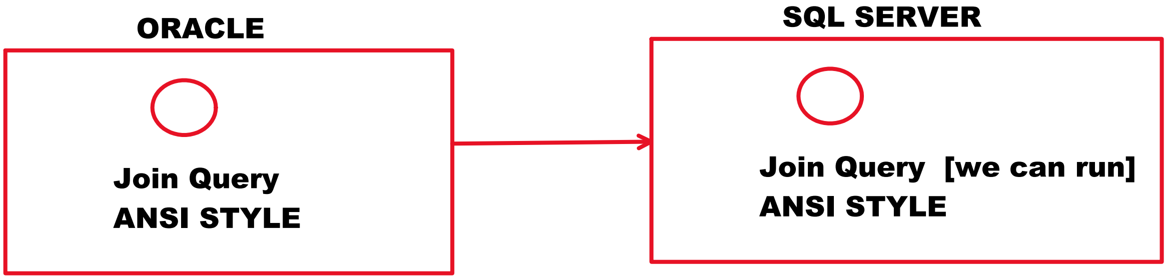
```
SELECT ename, sal, dname, loc
FROM emp, dept
WHERE emp.deptno=dept.deptno;

(or)

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

Above query degrades performance.
To improve performance, prefix column name with table name.

```
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 Join Query in 2 styles. They are:

- **ORACLE STYLE / NATIVE STYLE**
- **ANSI STYLE   => best way [portability]**



NOTE:

- **In ORACLE STYLE,**
  to separate 2 table names we use , [comma].
- **In ANSI STYLE,**
  to separate 2 table names we 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:

ENAME	SAL	DNAME	LOC

EMP e   **DEPT d**

ORACLE STYLE:

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

ANSI STYLE:

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

Display ALLEN record along with dept details.

ENAME	SAL	DNAME	LOC
ALLEN	..	..	..

ORACLE STYLE:

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

ANSI STYLE:

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

NOTE:

e.deptno=d.deptno	Join Condition
e.ename='ALLEN'	Filter Condition

First filter condition will be executed.
Then join condition will be executed.

to see execution plan:

```
SQL> SET AUTOTRACE ON EXPLAIN

SQL> <join query>

SQL> SET AUTOTRACE OFF
```

e.deptno=d.deptno

EMP e

EMPNO	ENAME	SAL	DEPTNO
7369	SMITH	..	20
7499	ALLEN	..	30
7521	WARD	..	30
7788	SCOTT	..	10
1001	A	..	
1002	B	..	

DEPT d

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

Display emp details along with dept details.
Display the emps who are working in NEW YORK.

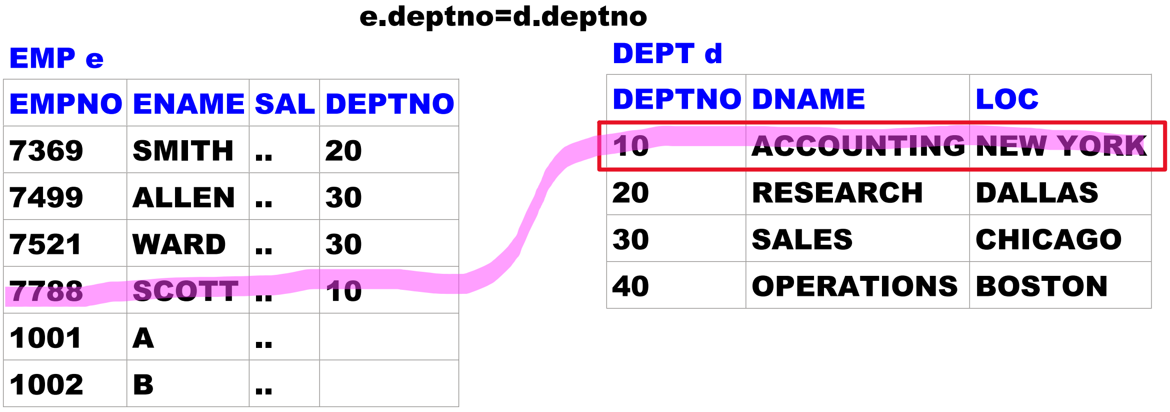
ENAME	SAL	DNAME	LOC
			NEW YORK

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

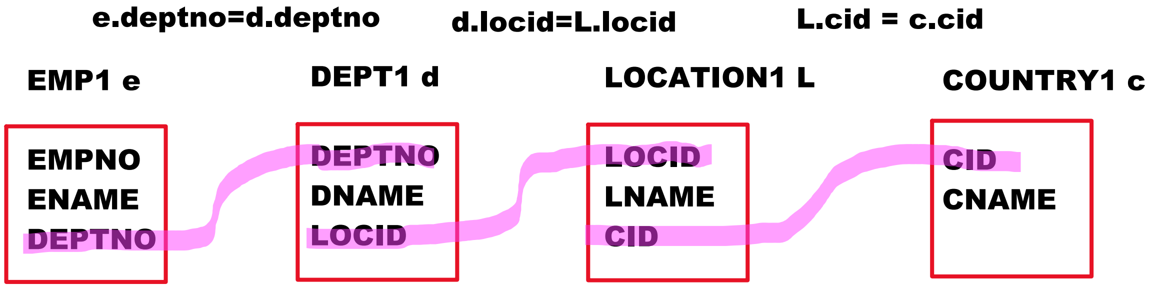
EMP e

EMPNO	ENAME	SAL	DEPTNO
7369	SMITH	..	20
7499	ALLEN	..	30
7521	WARD	..	30
7788	SCOTT	..	10
1001	A	..	
1002	B	..	

DEPT d

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

Retrieving data from 4 tables:



ENAME	DNAME	LNAME	CNAME
emp1 e	dept1 d	location1 L	country1 c

ORACLE STYLE:

```
SELECT e.ename, d.dname, L.Lname, c.cname
```

```
FROM emp1 e, dept1 d, Location1 L, country1 c
WHERE e.deptno=d.deptno AND
d.locid=L.locid AND
L.cid = c.cid;
```

ANSI STYLE:

```
SELECT e.ename, d.dname, L.Lname, c.cname
FROM emp1 e INNER JOIN dept1 d
ON e.deptno=d.deptno INNER JOIN  Location1 L
ON d.Locid=L.Locid INNER JOIN country1 c
ON L.cid = c.cid;
```

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:

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

Examples:

```
WHERE e.deptno > d.deptno
WHERE e.deptno < d.deptno
WHERE e.deptno != d.deptno
```

Example on Non-Equi Join:

```
WHERE e.sal BETWEEN s.losal AND s.hisal
```

EMP e

EMPNO	ENAME	SAL
1001	A	1800
1002	B	1300
1003	C	5000

SALGRADE s

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

ENAME	SAL	GRADE

emp e salgrade s

ORACLE STYLE:

```
SELECT e.ename, e.sal, s.grade
FROM emp e, salgrade s
WHERE e.sal BETWEEN s.losal AND s.hisal;
```

ANSI STYLE:

```
SELECT e.ename, e.sal, s.grade
FROM emp e INNER JOIN salgrade s
ON e.sal BETWEEN s.losal AND s.hisal;
```

Outer Join:

- **Inner Join = matched records only**
- **Outer join = matched + unmatched records**

- **Inner Join gives matched records only. To get unmatched records also we use Outer Join.**

- **Outer Join gives matched records and unmatched records.**

- **It has 3 sub types. They are:**
    - **Left Outer Join**
    - **Right Outer Join**
    - **Full Outer Join**

NOTE:
In ORACLE STYLE, based on join condition we can decide left table and right table.

WHERE e.deptno = d.deptno

emp e	Left table
dept d	Right table

WHERE d.deptno = e.deptno

dept d	Left table
emp e	Eight table

In ANSI STYLE, based on keyword we can decide left table and right table.

Example:
FROM emp e INNER JOIN dept d

emp e	Left table
dept d	Right table

FROM dept d INNER JOIN emp e

dept d	Left table
emp e	Eight table

Left Outer Join:

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

Example on Left Outer join:

EMP e

EMPNO	ENAME	SAL	DEPTNO
7369	SMITH	..	20
7499	ALLEN	..	30
7521	WARD	..	30
7788	SCOTT	..	10
1001	A	..	
1002	B	..	

unmatched records from emp

DEPT d

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

unmatched record from dept

Display emp details along with dept details.
Also display the emps to whom dept is not assigned.

ENAME	SAL	DNAME	LOC
SMITH	..	RESEARCH	DALLAS
A	..		

matched
unmatched from emp

ORACLE STYLE:

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

ANSI STYLE:

```
SELECT e.ename, e.sal, d.dname, d.loc
FROM emp e LEFT OUTER JOIN dept d
ON e.deptno = d.deptno;
```

Right Outer Join:

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

Example on Right Outer Join:

Display all emp details along with dept details.
Also display the depts which are not having emps:

ENAME	SAL	DNAME	LOC
SMITH	..	RESEARCH	DALLAS
		OPERATIONS	BOSTON

matched
unmatched from dept

ORACLE STYLE:

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

ANSI STYLE:

```
SELECT e.ename, e.sal, d.dname, d.loc
FROM emp e RIGHT OUTER JOIN dept d
ON e.deptno = d.deptno;
```

Full Outer Join:

- **Full Outer Join = matched + unmatched from left and right**
- **Full outer join gives matched records, unmatched records from left and right tables.**

- **In ANSI STYLE, use the keyword: FULL [OUTER] JOIN**
- **In ORACLE STYLE, we use UNION operator between Left outer join and Right outer join.**

In maths, SETS

A = {1,2,3,4,5}
B = {4,5,6,7,8}

A U B = {1,2,3,4,5,6,7,8}

Left Outer Join = matched + unmatched from left table
UNION
Right outer join = matched + unmatched from right table

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

Example on Full Outer Join:

Display emp details along with dept details.
Also display the emps to whom dept is not assigned.
Also display the depts which are not having emps.

ENAME	SAL	DNAME	LOC
SMITH	..	RESEARCH	DALLAS
A	..		
		OPERATIONS	BOSTON

matched
unmatched from emp
unmatched from dept

ORACLE STYLE:

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

ANSI STYLE:

```
SELECT e.ename, e.sal, d.dname, d.loc
FROM emp e FULL OUTER JOIN dept d
ON e.deptno=d.deptno;
```

Displaying unmatched records only:

- **Left Outer Join + condition = unmatched from left table**
- **Right outer join + condition = unmatched from right table**
- **Full outer join + conditions = um from left + um from right**

Left Outer Join + condition:

Display the emp records to whom dept is not assigned as following:

ENAME	DNAME
A	

unmatched from emp (left table)

ORACLE STYLE:

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

ANSI STYLE:

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

Right outer join + condition:

Display the depts which are not having emps as following:

ENAME	DNAME
	OPERATIONS

unmatched from dept (right)

ORACLE STYLE:

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

ANSI STYLE:

```
SELECT e.ename, d.dname
```

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

Full outer join + conditions:

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

ENAME	DNAME
A	
	OPERATIONS

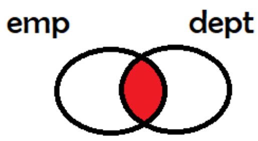
ORACLE STYLE:

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

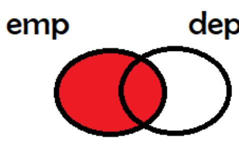
ANSI STYLE:

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

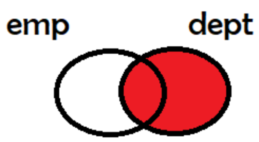
Venn diagrams of Joins:



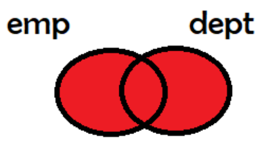
Inner Join



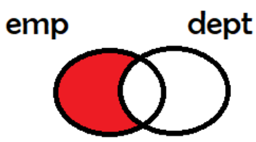
Left outer Join



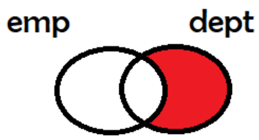
Right outer Join



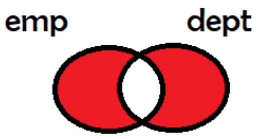
Full outer Join



left outer + condn



right outer + condn



full outer + condns

Self-Join:

- **If a table is joined to itself then it is called "Self-Join".**
- **In Self-Join, one record in a table will be joined with another record in same table.**
- **It can be also called as "Recursive Join".**

Example:

e.mgr = m.empno

emp e

empno	ename	sal	job	mgr
1001	A	25000	MANAGER	
1002	B	10000	CLERK	1001
1003	C	12000	ANALYST	1001
1004	D	30000	MANAGER	
1005	E	15000	SALESMAN	1004

emp m

empno	ename	sal	job	mgr
1001	A	25000	MANAGER	
1002	B	10000	CLERK	1001
1003	C	12000	ANALYST	1001
1004	D	30000	MANAGER	
1005	E	15000	SALESMAN	1004

Display emp details along with manager details:

EMP_NAME	EMP_SAL	MGR_NAME	MGR_SAL
B	10000	A	25000

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 emp details along with manager details.
Display the emps who are earning more than their manager:

EMP_NAME	EMP_SAL	MGR_NAME	MGR_SAL
	15000		20000

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 emp details along with manager details.
Display the emps who are working under BLAKE:

ORACLE STYLE:

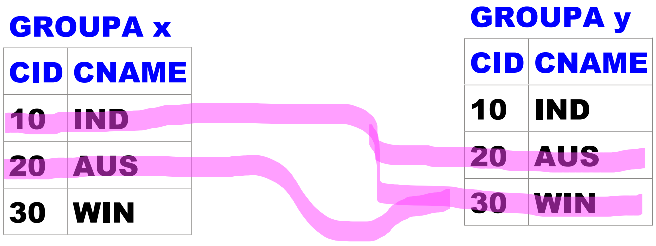
```
SELECT e.ename AS emp_name, e.sal AS emp_sal,
m.ename AS mgr_name, m.sal AS mgr_sal
FROM emp e, emp m
WHERE e.mgr=m.empno AND m.ename='BLAKE';
```

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 m.ename='BLAKE';
```

Example:

x.cid < y.cid



GROUPA x

CID	CNAME
10	IND
20	AUS
30	WIN

GROUPA y

CID	CNAME
10	IND
20	AUS
30	WIN

IND VS AUS
IND VS WIN
AUS VS WIN

ORACLE STYLE:

```
SELECT x.cname ||  ' VS ' || y.cname
FROM groupA x, groupA y
WHERE x.cid<y.cid;
```

ANSI STYLE:

```
SELECT x.cname ||  ' VS ' || y.cname
FROM groupA x INNER JOIN groupA y
```

```
ON x.cid<y.cid;
```

Cross Join / Cartesian Join:

- In Cross Join, Each record in one table will be joined with every record in another table.

- In cross join, don't write any join condition.

- In ANSI STYLE use the keyword: CROSS JOIN.

Example:

GROUPA a

cid	cname
10	IND
20	AUS
30	WIN

GROUPB b

cid	cname
40	ENG
50	PAK
60	BAN

```
IND VS ENG
IND VS PAK
IND VS BAN

AUS VS ENG
AUS VS PAK
AUS VS BAN

WIN VS ENG
WIN VS PAK
WIN VS BAN
```

```
SELECT a.cname || ' VS ' || b.cname
FROM groupA a, groupB b;
```

JOINS:

what is join?
JOIN => is an operation.
in this operation, it joins one table record with another table record based on join condn.

what is the purpose of join?
is used to retrieve data from multiple tables

Types of Joins:

Inner Join		**matched records only**
	Equi	**based on =, it performs join operation**
	Non-Equi	**based on other than =**
Outer Join		**matched + unmatched records**
	Left outer	**matched + um from left table**
	Right outer	**matched + um from right**
	Full outer	**matched + um from L and R**
Self-Join		**a table will be joined to itself**
Cross Join		**each record in 1 table will be joined with every record in another table**
Natural Join		**equi join without duplicate columns**

Natural Join:
equi join without duplicate columns

```
SELECT *
FROM emp e NATURAL JOIN dept d;
```

Tuesday, October 29, 2024 7:47 AM

EMP e

EMPNO	ENAME	SAL	DEPTNO
7369	SMITH	..	20
7499	ALLEN	..	30
7521	WARD	..	30
7788	SCOTT	..	10
1001	A	..	
1002	B	..	

DEPT d

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

Assignment

Friday, November 1, 2024 9:02 AM

EMPLOYEE

EMPID	ENAME	PID
1001	A	20
1002	B	20
1003	C	30
1004	D	
1005	E	

PROJECT

PID	PNAME	DURATION
10	X	..
20	Y	..
30	Z	..

Display emp details along with project details

ENAME	PNAME
A	Y
B	Y
C	Z

Equi Join

Display emp details along with project details.
Also display the emps who are on bench (who are not participating in any project development):

ENAME	PNAME
A	Y
B	Y
C	Z
D	
E	

employee => left table
Left outer join

matched

unmatched

Display emp details along with project details.
Also display the projects which are not assigned to any employee:

ENAME	PNAME
A	Y
B	Y
C	Z
X	

matched

unmatched

project => right table

right outer join

Example:

STUDENT

SID	SNAME	CID
1001	A	10
1002	B	10
1003	C	20
1005	D	
1006	E	

COURSE

CID	CNAME
10	JAVA
20	AWS
30	CPP

Display student details along with course details

SNAME	CNAME
A	JAVA
B	JAVA
C	AWS

matched
equi join

Display student details along with course details.
Also display the courses which are not having students:

SNAME	CNAME
A	JAVA
B	JAVA
C	AWS
	CPP

Sub Queries

Saturday, November 2, 2024 7:42 AM

Sub 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 filter condition value is unknown to find it we write Sub Query.
- Sub query must be written in parenthesis.
- Inner query must be SELECT only.
  It cannot be INSERT / UPDATE / DELETE.
- Outer query can be SELECT / INSERT / UPDATE / DELETE.

Types of Sub queries:

- Non-Correlated Sub query
  - Single Row Sub query
  - Multi row sub query
  - Inline view / Inline sub query
  - Scalar sub query
- Correlated sub query

Non-Correlated Sub Query:

- In this,
  First inner query gets executed.
  Then outer query gets executed.
  Inner query gets executed only 1 time.
- It has following sub types:
  - Single Row Sub query
  - Multi row sub query
  - Inline view / Inline sub query
  - Scalar sub query

Single Row Sub Query:

Syntax:

```
SELECT <column_list>
FROM <table_name>
WHERE <column> <operator> (<SELECT query>);
```

```
SELECT <column_list>
FROM <table_name>
WHERE <column> <operator> (<SELECT query>);
```

- **If sub query returns 1 row then it is called "Single Row Sub Query".**

Examples on Single row sub query:

Display the emp records who are earning more than BLAKE:

```
SELECT ename, sal
FROM emp
WHERE sal>(find BLAKE sal);
```

```
SELECT ename, sal
FROM emp
WHERE sal>(SELECT sal FROM emp
WHERE ename='BLAKE');
```

Display the emp records whose job title is same as SMITH:

```
SELECT ename, job, sal
FROM emp
WHERE job=(find SMITH job title);
```

```
SELECT ename, job, sal
FROM emp
WHERE job=(SELECT job FROM emp
WHERE ename='SMITH');
```

Find 2nd max salary:

```
3000
5000
4000
2500
1000
```

```
SELECT max(sal) FROM emp
WHERE sal<(find max sal);
```

```
3000
4000
2500
1000
```

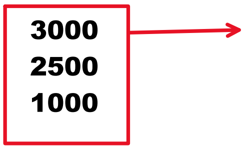
max sal => 4000
2nd max sal

```
SELECT max(sal) FROM emp
WHERE sal<(SELECT max(Sal) FROM emp);
```

Find 3rd max salary:

```
SELECT max(sal) FROM emp
WHERE sal < (find 2nd max sal);
```

3000
5000
4000
2500
1000



3000
2500
1000

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));
```

2nd max sal	1 SQ
3rd max sal	2 SQ
10th max sal	9 SQ

Display the emp record who is earning max sal:

max sal	8050

```
SELECT * FROM emp
WHERE sal=(find max sal);
```

```
SELECT * FROM emp
WHERE sal=(SELECT max(Sal) FROM emp);
```

Display the emp record who is earning 2nd max salary:

```
SELECT * FROM emp
WHERE sal=(find 2nd max sal);
```

```
SELECT * FROM emp
WHERE sal=(SELECT max(sal) FROM emp
WHERE sal<(SELECT max(sal) FROM emp));
```

Display seniors of BLAKE:

```
SELECT ename, hiredate
FROM emp
WHERE hiredate<(find BLAKE hiredate);
```

```
SELECT ename, hiredate
FROM emp
WHERE hiredate<(SELECT hiredate FROM emp
WHERE ename='BLAKE');
```

Display juniors of BLAKE:

```
SELECT ename, hiredate
FROM emp
WHERE hiredate>(find BLAKE hiredate);
```

```
SELECT ename, hiredate
FROM emp
WHERE hiredate>(SELECT hiredate FROM emp
WHERE ename='BLAKE');
```

Display most senior record:

```
SELECT * FROM emp
WHERE hiredate = (find min hiredate);
```

```
SELECT * FROM emp
WHERE hiredate = (SELECT min(hiredate)
FROM emp);
```

Display most junior record:

```
SELECT * FROM emp
WHERE hiredate = (find max hiredate);
```

```
SELECT * FROM emp
WHERE hiredate = (SELECT max(hiredate)
FROM emp);
```

Find the deptno which is spending max amount on salaries:

```
SELECT deptno
FROM emp
GROUP BY deptno
HAVING sum(Sal) = (find maximum sum of sal in all depts);
```

find maximum sum of sal in all depts:

```
SELECT max(sum(sal)) FROM emp
GROUP BY deptno;
```

```
SELECT deptno
FROM emp
GROUP BY deptno
HAVING sum(Sal) = (SELECT max(sum(sal)) FROM emp
GROUP BY deptno);
```

Find the dept name which is spending max amount on salaries:

```
SELECT dname FROM dept
WHERE deptno=(find the deptno which spending max amt);
```

```
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));
```

Multi row Sub Query:

- **If sub query returns multiple rows then it is called "Multi Row Sub Query".**
- **For multi row sub query we use following operators:**
  - **IN**
  - **ALL**
  - **ANY**

  Syntax:

```
SELECT <column_list>
FROM <table_name>
WHERE <column> <operator> (<select query>);
```

Display the emp records whose job titles are same as SMITH and BLAKE job titles:

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 for multi value comparison.**

Example:

sal>ALL(6000,4000) if sal > all list values, condn is TRUE SAL ------- 8000 8000>ALL(6000,4000) => T 5000 5000>ALL(6000,4000) => F 2000 2000>ALL(6000,4000) => F	sal>6000 AND sal>4000

5500 5500>ALL(6000,4000) => F 9000 9000>ALL(6000,4000) => T	

Display the emp records whose salaries are more than BLAKE and WARD:

BLAKE	**6000**
WARD	**4000**

```
SELECT ename, sal
FROM emp
WHERE sal>ALL(find BLAKE sal AND WARD sal);
```

```
SELECT ename, sal
FROM emp
WHERE sal>ALL(SELECT sal FROM emp
WHERE ename IN('BLAKE','WARD'));
```

ANY:

- **it is used for multi value comparison.**

Example:

sal>ANY(6000, 4000) SAL ------- 8000 8000>ANY(6000,4000) T 5000 5000>ANY(6000,4000) T 2000 2000>ANY(6000,4000) F 5500 5500>ANY(6000,4000) T 9000 9000>ANY(6000,4000) T	sal>6000 OR sal>4000

Display the emp records whose salaries are more than BLAKE sal or WARD sal:

```
SELECT ename, sal
FROM emp
WHERE sal>ANY(find BLAKE and WARD salaries);
```

```
SELECT ename, sal
FROM emp
WHERE sal>ANY(SELECT sal FROM emp
WHERE ename IN('BLAKE','WARD'));
```

NOTE:

- **when we don't know filter condition to find it we write single row sq (or) multi row sq.**
- **to control execution order of clauses we write inline view**

Inline View:

- **If sub query is written in FROM clause then it is called "Inline View".**
- **To control execution order of clauses we write sub query in "FROM" clause.**
- **It acts like table.**

Syntax:

```
SELECT <column_list>
FROM (<sub query>)
WHERE <condition>;
```

Examples on Inline View:

Find 3rd max sal:

```
SELECT ename, sal,
dense_rank() OVER(ORDER By sal DESC) AS rank
FROM emp
WHERE rank=3;
```

Output:

```
ERROR: RANK invalid identifier
```

```
SELECT sal FROM emp
```

```
SELECT DISTINCT sal
FROM (SELECT ename, sal,
dense_rank() OVER(ORDER By sal DESC) AS rank
FROM emp)
WHERE rank=3;
```

Find 5th max sal:

```
SELECT DISTINCT sal
FROM (SELECT ename, sal,
dense_rank() OVER(ORDER By sal DESC) AS rank
FROM emp)
WHERE rank=5;
```

Find nth max sal:

```
SELECT DISTINCT sal
FROM (SELECT ename, sal,
dense_rank() OVER(ORDER By sal DESC) AS rank
FROM emp)
WHERE rank=&n;
```

Find top 3 salaries:

```
SELECT DISTINCT sal
FROM (SELECT ename, sal,
dense_rank() over(order by sal desc) as rank
FROM emp)
WHERE rank<=3;
```

ROWNUM:

- **It is a pseudo column.**
- **It is used to apply row numbers to records**

Examples on ROWNUM:

Display all emp records.
Apply row numbers to them:

```
SELECT rownum AS sno, ename, sal
FROM emp;
```

Display all managers records.
Apply row numbers to them:

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

Display 3rd record from emp:

```
SELECT *
FROM (SELECT rownum as rn, empno, ename, sal
FROM emp)
WHERE rn=3;
```

Display 3rd row, 7th row and 11th row from emp:

```
SELECT *
FROM (SELECT rownum as rn, empno, ename, sal
FROM emp)
WHERE rn IN(3,7,11);
```

Display 6th row to 10th row:

```
SELECT *
FROM (SELECT rownum as rn, empno, ename, sal
FROM emp)
WHERE rn BETWEEN 6 AND 10;
```

Display even numbered rows:

```
SELECT *
```

```
FROM (SELECT rownum as rn, empno, ename, sal
FROM emp)
WHERE MOD(rn,2)=0;
```

Scalar Sub Query:

- If a sub query is written in SELECT clause then it is called "Scalar Sub Query".
- It acts like column.

Syntax:

```
SELECT (<sub query>), ....
FROM <table_name>
WHERE <condition>;
```

Examples:

Display no of records in emp table and dept table.

EMP	DEPT
14	4

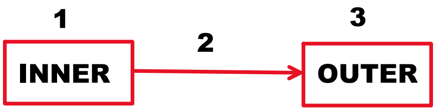
```
SELECT (SELECT count(*) FROM emp) AS emp,
(SELECT count(*) FROM dept) AS dept
FROM dual;
```

Find each dept share in salaries:

DEPTNO	SUM_OF_SAL	AMOUNT	PER
10	17929.1	70955.45	17929.1*100/70955.45 = 25.2681
20	23486.05	70955.45	23486.05*100/70955.45 = 33.0997
30	29540.3	70955.45	29540.3*100/70955.45 = 41.6322

```
SELECT deptno, sum(sal) AS sum_of_sal,
(SELECT sum(sal) FROM emp) AS amount,
TRUNC(sum(sal)*100/(SELECT sum(Sal) FROM emp),2) AS per
FROM emp
GROUP BY deptno
ORDER BY 1;
```

Non-Correlated Sub Query:



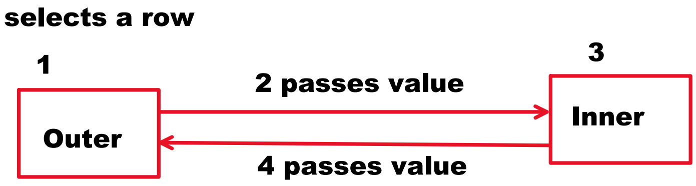
Execution process:
1. First Inner query gets executed.
2. Inner query passes value to Outer query.
3. Outer query gets executed.

Inner query gets executed only 1 time.

Correlated Sub Query:

- **Outer query passes value to inner query then it is called "Correlated Sub Query".**
- **In this, Inner gets executed multiple times.**

Execution Process:



selects a row

1

Outer

2 passes value

3

Inner

4 passes value

5. condition => T selects the 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 condn is TRUE selects the row.**

These 5 steps will be executed repeatedly for every row selected by outer query.

Examples on Correlated Sub Query:

Display the emp records who are earning more than their dept's avrg salary:

emp sal > emp dept's avrg sal

EMP e

EMPNO	ENAME	DEPTNO	SAL
1001	A	10	10000
1002	B	10	20000
1003	C	20	30000
1004	D	20	10000

deptno	avg(Sal)
10	15000
20	20000

ENAME	DEPTNO	SAL

```
SELECT 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

deptno	avg(Sal)
10	15000

EMP e

EMPNO	ENAME	DEPTNO	SAL
1001	A	10	10000
1002	B	10	20000
1003	C	20	30000
1004	D	20	10000

deptno	avg(Sal)
10	15000
20	20000

ENAME	DEPTNO	SAL
B	10	20000
C	20	30000

Display the emp records who are earning max salary in their dept:

emp sal = emp dept's max sal

ENAME	DEPTNO	SAL

```
SELECT ename, deptno, sal
FROM emp e
WHERE sal = (SELECT max(Sal) FROM emp
WHERE deptno=e.deptno);
```

EMP e

EMPNO	ENAME	DEPTNO	SAL
1001	A	10	10000
1002	B	10	20000
1003	C	20	30000
1004	D	20	10000

deptno	max(Sal)
10	20000
20	30000

ENAME	DEPTNO	SAL
B	10	20000
C	20	30000

Assignment:
Display most senior in each dept:

ENAME	DEPTNO	HIREDATE

emp hiredate = emp dept's min hiredate

EXISTS:

- **If sub query selects rows then EXISTS returns true.**
- **If sub query does not select rows then EXISTS returns FALSE.**

Display the dept names which are having emps:

DEPT d

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

EMP e

EMPNO	ENAME	SAL	DEPTNO
7369	SMITH	..	20
7499	ALLEN	..	30
7521	WARD	..	30
7788	SCOTT	..	10

30	SALES	CHICAGO
40	OPERATIONS	BOSTON

7521	WARD	..	30
7788	SCOTT	..	10
1001	A	..	
1002	B	..	

```
SELECT dname
FROM dept d
WHERE exists(SELECT * FROM emp WHERE deptno=d.deptno);
```

DNAME
ACCOUNTING
RESEARCH
SALES

Display the dept names which are not having emps:

```
SELECT dname
FROM dept d
WHERE not exists(SELECT * FROM emp WHERE deptno=d.deptno);
```

ROWID

Wednesday, November 6, 2024 7:52 AM

ROWID:

- **ROWID is a pseudo column.**
- **It is used to get address of the row.**
- **It is useful to deal with duplicate records.**

Examples:

Display all columns and rows of dept table along with row ids:

```
SELECT rowid, d.* FROM dept d;
```

Example:

EMPLOYEE

	EMPID	ENAME	SAL
AAA	1001	A	10000
AAB	1001	A	10000

delete duplicate record:

```
DELETE FROM employee
WHERE rowid=<rowid>;

DELETE FROM employee
WHERE rowid='AAATsRAAHAAAAb/AAE';
```

EMPLOYEE

EMPID	ENAME	SAL
1001	A	10000

1001	A	10000
1001	A	10000
1001	A	10000
1001	A	10000
1001	A	10000
1001	A	10000

delete duplicate records:

Correlated sub query

video link:
bit.ly/deletedups

CONSTRAINTS

Wednesday, November 6, 2024 8:09 AM

CONSTRAINTS:

- **CONSTRAINT => restrict / control / limit**

GOAL:

- **CONSTRAINT is used to restrict the user from entering invalid data.**

CHECK(gender IN('M','F'))
GENDER

M
F
Z invalid data => error

CONSTRAINTS:

- **CONSTRAINT is a rule that is applied on a column.**
- **CONSTRAINT is used to restrict the user from entering invalid data.**
- **It is used to implement data integrity feature.**
- **Maintaining accurate and quality data is called "data integrity".**
- **Using CONSTRAINT, we can maintain quality data and accurate data.**

ORACLE SQL provides following Constraints:

- **Primary Key**
- **Not Null**
- **Unique**
- **Check**
- **Default**
- **References [Foreign Key]**

Primary Key:

- **A column which has unique values on that we apply PRIMARY KEY.**
- **It does not accept duplicates.**
- **It does not accept nulls.**
- **When value is mandatory and it should not be duplicated then use PRIMARY KEY.**

Example:

EMPLOYEE

PK

EMPID	ENAME	JOB	SAL

1001	RAVI	CLERK	12000
1002	KIRAN	CLERK	10000
1003	RAVI	SALESMAN	12000
ERROR: null	SRAVAN	MANAGER	20000
1001 ERROR: duplicate	SAI	CLERK	15000

Example:

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

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

INSERT INTO t1 VALUES(null);   --error
INSERT INTO t1 VALUES(1);      --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	
EMPID	ENAME	SAL
1001	KIRAN	10000
1002	SAI	12000
1003	KIRAN	15000
1004	error: null	20000

Example:

```
CREATE TABLE t2
(
f1 INT NOT NULL
);

INSERT INTO t2 VALUES(1);
INSERT INTO t2 VALUES(1);
INSERT INTO t2 VALUES(2);
INSERT INTO t2 VALUES(null);  --ERROR
```

Unique:

- It does not accept duplicates.
- It accepts nulls.
- When value is optional and that should not be duplicated then use UNIQUE.

Example:

CUSTOMER

UNIQUE

CUSTID	CNAME	MAIL_ID
1001	SAI	sai@gmail.com
1002	KIRAN	□
1003	RAJU	sai@gmail.com ERROR: duplicate

Example:

```
CREATE TABLE t3
(
f1 INT UNIQUE
);

INSERT INTO t3 VALUES(1);
INSERT INTO t3 VALUES(1);   --error
INSERT INTO t3 VALUES(null);
INSERT INTO t3 VALUES(null);
INSERT INTO t3 VALUES(null);
```

CONSTRAINT	DUPLICATES	NULLS
PRIMARK KEY	NO	NO
NOT NULL	YES	NO
UNIQUE	NO	YES

PRIMARY KEY = UNIQUE + NOT NULL

CHECK:

- It is used to apply our own condition on column.

Example:

```
CHECK(gender IN('M','F'))
GENDER
----------------
F
M
Z    error
```

Example:

```
CREATE TABLE t4
(
gender CHAR CHECK(gender IN('M','F'))
);

INSERT INTO t4 VALUES('M');
INSERT INTO t4 VALUES('F');
INSERT INTO t4 VALUES('Z');   --error
```

Default:

- It is used to apply default value to column.
- When for most of records value is same then it's better set it as default value.

Example:

DEFAULT 20000

SID	SNAME	FEE
1001	A	20000
1002	B	20000
1003	C	20000
1004	D	10000

Example:

```
CREATE TABLE t5
(
sid NUMBER(4),
fee NUMBER(7,2) DEFAULT 20000
);

INSERT INTO t5 VALUES(1001);
error: not enough values

INSERT INTO t5(sid) VALUES(1001);
INSERT INTO t5(sid) VALUES(1002);
INSERT INTO t5(sid) VALUES(1003);

INSERT INTO t5 VALUES(1004, 10000);
```

REFERENCES [FOREIGN KEY]:

- Foreign Key accepts Primary key values of another table.
- It is used to establish relationship between 2 tables.

Example:

Example:

COURSE

PK

CID	CNAME
10	JAVA
20	PYTHON
30	C#

STUDENT

FK REFERENCES COURSE(CID)

SID	SNAME	CID
1001	A	10
1002	B	10
1003	C	30
1004	D	20
1005	E	90 ERROR

PRODUCTS

PID	PNAME	PRICE
1001	A	..
1002	B	..

CUSTOMERS

CUSTID	CNAME	MOBILE
123456	XYZ	..

ORDERS

ORDERID	PID FK REFERENCES PRODUCTS(PID)	CUSTID FK REFERENCES customers(custid)
67890012		

Examples on Constraints:

Exampe-1:

STUDENT

SID	SNAME	M1

SID	don't accept dups and nulls	PK
SNAME	don't accept nulls	NOT NULL
M1	must be b/w 0 to 100	CHECK

```
CREATE TABLE student
(
   sid NUMBER(4) PRIMARY KEY,
   sname VARCHAR2(10) NOT NULL,
   m1 NUMBER(3) CHECK(m1 BETWEEN 0 AND 100)
);
```

sai@gmail.com

NOTE:

A table can have only 1 PK.

Example-2:

USERINFO

USERID	UNAME	PWD

USERID	don't accept dups and nulls	PK
uname	don't accept dups and nulls	UNIQUE NOT NULL
pwd	min 8 chars	CHECK

```
CREATE TABLE userinfo
(
userid NUMBER(4) PRIMARY KEY,
uname VARCHAR2(15) UNIQUE NOT NULL,
pwd VARCHAR2(20) CHECK(length(pwd)>=8)
);
```

Example-3:

DEPT1

PK

DEPTNO	DNAME
10	HR
20	ACCOUNTS
30	SALES

EMP1

PK FK REFERENCES DEPT1(deptno)

EMPNO	ENAME	DEPTNO
1001	A	30
1002	B	80 ERROR

```
CREATE TABLE dept1
(
deptno NUMBER(2) PRIMARY KEY,
dname VARCHAR2(10)
);

CREATE TABLE emp1
(
empno NUMBER(4) PRIMARY KEY,
ename VARCHAr2(10),
deptno NUMBER(2) REFERENCES dept1(deptno)
);
```

NOTE:

PK column data type and FK column data type must be same.

PK does not accept dups and nulls.
FK accepts dups and nulls.

Example-4:

EMPLOYEE20

EMPID	ENAME	GENDER	SAL	CNAME

empid	don't accept dups and nulls	PK
ename	don't accept nulls	NOT NULL
gender	accept M or F	CHECK
sal	min 5000	CHECK
cname	default value TCS	DEFAULT

```
CREATE TABLE employee20
(
empid NUMBER(4) PRIMARY KEY,
ename VARCHAR2(10) NOT NULL,
gender CHAR CHECK(gender IN('M','F')),
sal NUMBER(8,2) CHECK(sal>=5000),
cname VARCHAR2(3) DEFAULT 'TCS'
);
```

```
INSERT INTO employee20(empid, ename, gender, sal)
VALUES(1001, 'A', 'M', 8000);
```

Naming Constraints

Thursday, November 7, 2024 9:00 AM

Syntax of creating table:

```
CREATE TABLE <name>
(
   <field_name> <data_type> [CONSTRAINT <con_name> <con_type> ,
   <field_name> <data_type> CONSTRAINT <con_name> <con_type> ,
   .
   .]
)
```

Naming Constraints:

- **We can give names to constraints.**
- **As a developer we have to give constraint name.**
  If we don't give constraint name implicitly ORACLE gives a constraint name.
- **To identify constraint uniquely in entire DB this name is useful.**
- **To disable or enable or drop the constraint we use this constraint name.**

Example:

STUDENT12

SID	SNAME	M1

SID	PK	constraint name => c1
SNAME		
M1	CHECK	constraint name => c2

```
CREATE TABLE student12
(
sid NUMBER(4) CONSTRAINT c1 PRIMARY KEY,
sname VARCHAR2(10),
m1 NUMBER(3) CONSTRAINT c2 CHECK(m1 BETWEEN 0 AND 100)
);
```

Friday, November 8, 2024 7:42 AM

CONSTRAINT can be applied at 2 levels.
They are:

- **Column Level Constraint**
- **Table Level Constraint**

Column Level Constraint:

- **If constraint is defined in column definition then it is called "Column Level Constraint".**
- **All 6 constraints can be applied at column level.**
  [PK, NOT NULL, UNIQUE, CHECK, DEFAULT, FK]

Example:

```
CREATE TABLE student12
(
sid NUMBER(4) CONSTRAINT c1 PRIMARY KEY,
sname VARCHAR2(10),
m1 NUMBER(3) CONSTRAINT c2 CHECK(m1 BETWEEN 0 AND 100)
);
```

Table Level Constraint:

- **If constraint is defined after defining all columns then it is called "Table Level Constraint".**
- **We can apply only 4 constraints at table level.**
  [PK, UNIQUE, CHECK, FK]

```
CREATE TABLE student13
(
sid NUMBER(4),
sname VARCHAR2(10),
m1 NUMBER(3),
CONSTRAINT c3 PRIMARY KEY(sid),
CONSTRAINT c4 CHECK(m1 BETWEEN 0 AND 100)
);
```

Applying Foreign Key at Table Level:

COURSE1
PK

CID	CNAME
10	JAVA

STUDENT1
FK REFERENCES COURSE1(cid)

SID	SNAME	CID
1001	A	20

20	PYTHON
30	ORACLE

1001	A	30
1002	B	30

```
CREATE TABLE course1
(
cid NUMBER(2),
cname VARCHAR2(10),
CONSTRAINT c5 PRIMARY KEY(cid)
);
```

```
CREATE TABLE student1
(
sid NUMBER(4),
sname VARCHAR2(10),
cid NUMBER(2),
CONSTRAINT c6  FOREIGN KEY(cid) REFERENCES course1(cid)
);
```

Why Table Level?

2 reasons:

- **to apply combination of columns as constraint**
- **to use another column name in constraint**

applying combination of columns as constraint:

STUDENT15
PK(sid, subject)

SID	SNAME	SUBJECT	MARKS
1001	A	SUB1	70
1001	A	SUB2	80
1002	B	SUB1	80
1002	B	SUB2	55
1001		SUB1 error: duplicate	
null error	..	..	..
		null error	

```
CREATE TABLE student15
(
sid NUMBER(4),
sname VARCHAR2(10),
subject CHAR(4),
marks NUMBER(2),
CONSTRAINT c10 PRIMARY KEY(sid, subject)
);
```

Composite Primary Key:
If PK is applied on combination of columns then it is called "Composite Primary Key".

Using another column name in constraint:

Example:

MAN_DATE	MANUFATURED DATE
EXP_DATE	EXPIRY DATE

PRODUCTS

CHECK(exp_date>man_date)

PID	PNAME	MAN_DATE	EXP_DATE
1001	A	8-NOV-24	8-Sep-23 error

```
CREATE TABLE products
(
pid NUMBER(4),
pname VARCHAR2(10),
man_date DATE,
exp_date DATE,
CONSTRAINT c11 CHECK(exp_date>man_date)
);
```

Altering Constraints

Friday, November 8, 2024 8:41 AM

ALTER:

ALTER command is used to change structure of table.

Using ALTER command we can:

- **Add the columns**
- **Rename the columns**
- **Drop the columns**
- **Modify data types**
- **Modify field sizes**

- **Add the constraints**
- **Rename the constraints**
- **Disable the constraints**
- **Enable the constraints**
- **Drop the constraints**

Syntax of ALTER command:

```
ALTER TABLE <table_name> [ADD CONSTRAINT <con_name> <con_type>(<column>)]
                         [MODIFY(<field_definitions>)]
                         [RENAME CONSTRAINT <old_name> TO <new_name>]
                         [DISABLE CONSTRAINT <con_name>]
                         [ENABLE CONSTRAINT <con_name>]
                         [DROP CONSTRAINT <con_name>];
```

field definition:

```
<col_name> <data_type> CONSTRAINT <con_name> <con_type>
```

Example:

STUDENT

SID	SNAME	M1

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

Add PK to sid:

```
ALTER TABLE student ADD CONSTRAINT c20 PRIMARY KEY(sid);
(or)
ALTER TABLE student MODIFY sid CONSTRAINT c20 PRIMARY KEY
```

NOTE:
Using ADD CONSTRAINT, we can add table level constraints (4 constraints)
Using MODIFY, we can add column level constraints (6 constraints)

Add not null to sname:

```
ALTER TABLE student MODIFY sname CONSTRAINT c21 NOT NULL;
```

Add check constraint to m1:

```
ALTER TABLE student ADD CONSTRAINT c22 CHECK(m1 BETWEEN 0 AND 100);
```

Rename PK constraint c20 to z:

```
ALTER TABLE student RENAME CONSTRAINT c20 TO z;
```

Disable PK z:

```
ALTER TABLE student DISABLE CONSTRAINT z;
```

Enable PK z:

```
ALTER TABLE student ENABLE CONSTRAINT z;
```

Drop PK z:

```
ALTER TABLE student DROP CONSTRAINT z;
```

user_constraints

Saturday, November 9, 2024 7:47 AM

user_consraints:

- **it is a system table / built-in table / readymade table.**
- **it maintains all constraints info.**

to see constraints info:

```
DESC user_constraints

SELECT table_name, constraint_name, constraint_type
FROM user_constraints;
```

to see constraints list of STUDENT12 table:

```
SELECT table_name, constraint_name, constraint_type
FROM user_constraints
WHERE table_name='STUDENT12';
```

SET OPERATORS

Saturday, November 9, 2024 7:55 AM

In maths, SETS:

A = {1,2,3,4,5}
B = {4,5,6,7,8}

A U B = {1,2,3,4,5,6,7,8} = 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}
It gives specific elements from A.
Element should be available in A only and should not be available in B
It gives all elements from A set except common elements

B M A = {6,7,8}
It gives specific elements from B.
It gives all elements from B set except common elements

SET OPERATORS:

- **SET OPERATOR is used to combine result of 2 select queries.**

  Syntax:
  ```
  <SELECT query>
          <SET OPERATOR>
  <SELECT query>
  ```

- **ORACLE SQL provides following SET OPERATORS:**
  - **UNION**
  - **UNION ALL**
  - **INTERSECT**
  - **MINUS**

UNION:

- **It combines result of 2 select queries without duplicates.**
- **it does not give duplicates.**
- **it is slower. Because, it checks for duplicates.**

UNION ALL:

- **It combines result of 2 select queries including duplicates.**
- **it gives duplicates.**
- **it is faster. Because, It does not checks for duplicates.**

INTERSECT:

- **It gives common records from result of 2 select queries.**

MINUS:

- **It gives specific records from first select query result.**

Examples on set operators:

CRICKET

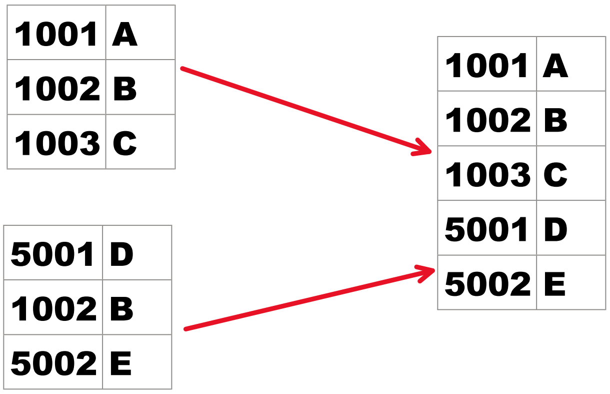
SID	SNAME
1001	A
1002	B
1003	C

FOOTBALL

SID	SNAME
5001	D
1002	B
5002	E

Display the students who are participating in CRICKET and FOOTBALL:

```
SELECT * FROM cricket
UNION
SELECT * FROM football;
```



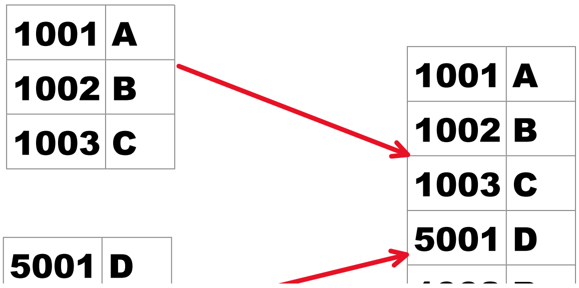
1001	A
1002	B
1003	C

5001	D
1002	B
5002	E

1001	A
1002	B
1003	C
5001	D
5002	E

Display the students who are participating in CRICKET and FOOTBALL including duplicates:

```
SELECT * FROM cricket
UNION ALL
SELECT * FROM football;
```



1001	A
1002	B
1003	C

5001	D

1001	A
1002	B
1003	C
5001	D

UNION ALL
SELECT * FROM football;

5001	D
1002	B
5002	E

1003	C
5001	D
1002	B
5002	E

Display the students who are participating CRICKET and FOOTBALL both:

```
SELECT * FROM cricket
INTERSECT
SELECT * FROM football;
```

1001	A
1002	B
1003	C

5001	D
1002	B
5002	E

1002	B

Display the students who are participating only in CRICKET and those students should not be participated in football:

```
SELECT * FROM cricket
MINUS
SELECT * FROM football;
```

1001	A
1002	B
1003	C

5001	D
1002	B
5002	E

1001	A
1003	C

Display the students who are participating only in FOOTBALL and those students should not be participated in cricket:

```
SELECT * FROM football
MINUS
SELECT * FROM cricket;
```

5001	D
1002	B
5002	E

1001	A

5001	D
5002	E

SELECT * FROM cricket;

1001	A
1002	B
1003	C

Example:

DEPTNO 20	DEPTNO 30
CLERK	CLERK
MANAGER	MANAGER
ANALYST	SALESMAN

Display the job titles offered by deptno 20 and 30:

```
SELECT job FROM emp WHERE deptno=20
UNION
SELECT job FROM emp WHERE deptno=30;
```

Display common job titles offered by deptno 20 and 30:

```
SELECT job FROM emp WHERE deptno=20
INTERSECT
SELECT job FROM emp WHERE deptno=30;
```

Display specific job titles of deptno 20 and those should not be offered by deptno 30:

```
SELECT job FROM emp WHERE deptno=20
MINUS
SELECT job FROM emp WHERE deptno=30;
```

Display specific job titles of deptno 30 and those should not be offered by deptno 20:

```
SELECT job FROM emp WHERE deptno=30
MINUS
SELECT job FROM emp WHERE deptno=20;
```

Example:

emp_us

EMPID	ENAME
1001	A
1002	B
1003	C

emp_IND

EMPID	ENAME
5001	D
5002	E
5003	F

Display the emps who are working in US and IND:

```
SELECT * FROM emp_us
UNION
SELECT * FROM emp_IND;
```

1001	A
1002	B
1003	C
5001	D
5002	E
5003	F

Rules of SET OPERATORS:

- **number of columns in both select queries must be same.**

  ```
  SELECT sid, sname FROM cricket
  UNION
  SELECT sid FROM football;
  ```

  Output:
  ERROR

- **Corresponding columns data types in both select queries must be same.**

  Example:

  ```
  SELECT sid, sname FROM student
  UNION
  SELECT sname, sid FROM cricket;
  ```

Output:
ERROR

Differences b/w UNION [set operator] and JOIN:

emp

empno	ename	sal	deptno

dept

deptno	dname	loc

ename	dname

UNION	• it combines rows • it is used for horizontal merging • it is applied on similar structures
JOIN	• it combines columns • it is used for vertical merging • it is applied on dissimilar structures

SQL Commands
Built-In Functions
Clauses
Joins
Sub queries
Constraints
Set operators

Copying Table

Saturday, November 9, 2024 9:02 AM

Copying Table:
Copying Table means,
creating new table from existing table.

IN this, a new table will be created with select query result.

Syntax:

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

Example:
Create exact copy of emp table with the name emp1:

EMP

8 cols
15 rows

EMP1

8 cols
15 rows

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

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

EMP

8 cols
15 rows

EMP2

4 cols => empno, ename, job, sal
managers

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

Copying table structure

Saturday, November 9, 2024 9:10 AM

Syntax:

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

false conditions

```
WHERE 1=2
      'A'='B'
      500 = 600
```

false_condition => does not select rows

Create a new table with emp table structure with the name emp3:

EMP
8 cols
15 rows

EMP30
8 cols
no rows

```
CREATE TABLE emp30
AS
SELECT * FROM emp
WHERE 1=2;
```

Copying Records

Monday, November 11, 2024 7:46 AM

Copying Records:

Syntax:

```
INSERT INTO <table_name>[(<column_list>)]
<SELECT QUERY>;
```

Example:

EMP

8 cols
14 rows

copy

EMP10

8 columns
no rows

Copy emp table all rows to emp10:

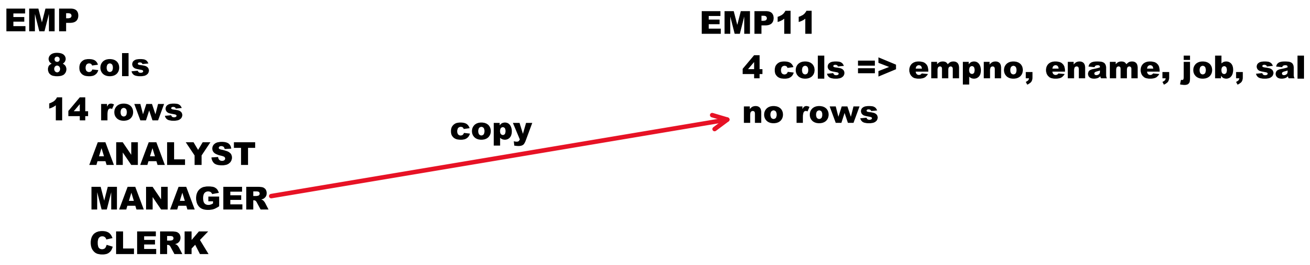
creating emp10 table from existing table emp without rows:

```
CREATE TABLE emp10
AS
SELECT * FROM emp WHERE 1=2;
```

Copy emp table all rows to emp10:

```
INSERT INTO emp10
SELECT * FROM emp;
```

Example:



EMP

8 cols
14 rows
ANALYST
MANAGER
CLERK

copy

EMP11

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

Copy all managers records to emp11 table:

creating emp11 table from existing table emp with 4 columns without rows:

EMPNO	ENAME	JOB	SAL

```
CREATE TABLE emp11
AS
SELECT empno, ename, job, sal
FROM emp
WHERE 1=2;
```

Copy all managers records to emp11:

```
INSERT INTO emp11
SELECT empno, ename, job, sal
FROM emp WHERE job='MANAGER';
```

INSERT ALL

Monday, November 11, 2024 8:04 AM

INSERT ALL:

- **Introduced in oracle 9i.**
- **It command is used to copy ne table data to multiple tables.**
- **It avoids of writing multiple INSERT commands.**
- **It is mainly used to perform ETL operations.**
  ETL => Extract Transfer Load

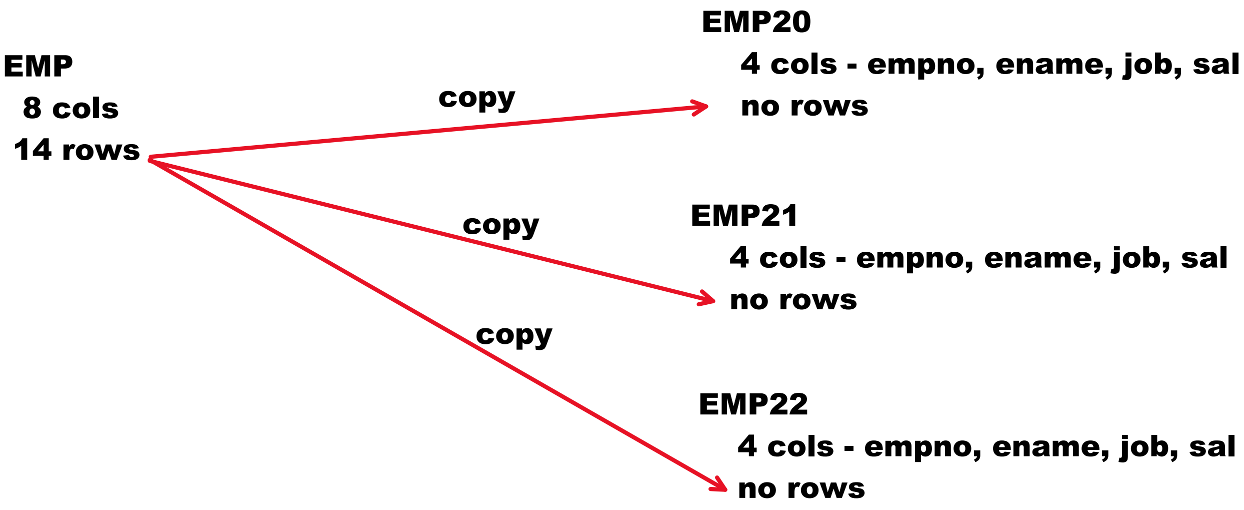
- **INSERT ALL can be used in 2 ways. They are:**
  - **Unconditional INSERT ALL**
  - **Conditional INSERT ALL**

Unconditional INSERT ALL:

Syntax:

```
INSERT ALL
INTO <table_name>(col_list>) VALUES(<value_list>)
INTO <table_name>(col_list>) VALUES(<value_list>)
.
.
<select query>;
```

Example on Unconditional INSERT ALL:



EMP
8 cols
14 rows

copy

EMP20
4 cols - empno, ename, job, sal
no rows

copy

EMP21
4 cols - empno, ename, job, sal
no rows

copy

EMP22
4 cols - empno, ename, job, sal
no rows

create emp20, emp21, emp22:

```
CREATE TABLE emp20
AS
SELECT empno, ename, job, sal
FROM emp
WHERE 1=2;

CREATE TABLE emp21
AS
SELECT empno, ename, job, sal
FROM emp
WHERE 1=2;

CREATE TABLE emp22
AS
SELECT empno, ename, job, sal
FROM emp
WHERE 1=2;
```

copy emp table all rows, 4 cols to emp20, emp21, emp22:

```
INSERT ALL
INTO emp20 VALUES(empno, ename, job, sal)
INTO emp21 VALUES(empno, ename, job, sal)
INTO emp22 VALUES(empno, ename, job, sal)
SELECT empno, ename, job, sal FROM emp;
```

Output:

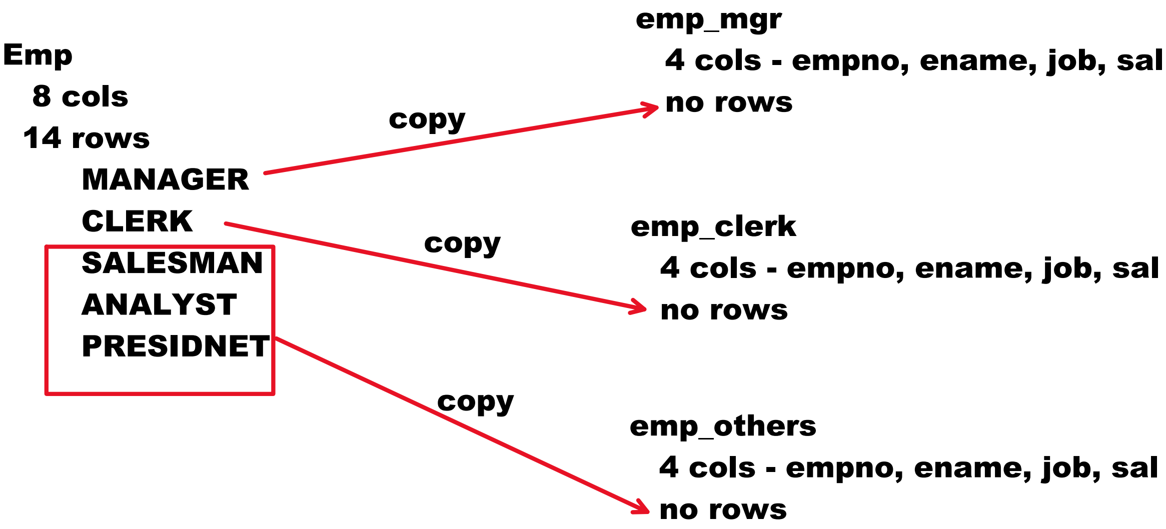
```
42 rows created.
```

Conditional INSERT ALL:

Syntax:

```
INSERT ALL
WHEN <condition1> THEN
   INTO <table_name>(col_list>) VALUES(<value_list>)
WHEN <codition2> THEN
   INTO <table_name>(col_list>) VALUES(<value_list>)
.
.
ELSE
   INTO <table_name>(col_list>) VALUES(<value_list>)
<select query>;
```

Example on Conditional INSERT ALL:



Emp
8 cols
14 rows
MANAGER
CLERK
SALESMAN
ANALYST
PRESIDNET

copy

emp_mgr
4 cols - empno, ename, job, sal
no rows

copy

emp_clerk
4 cols - empno, ename, job, sal
no rows

copy

emp_others
4 cols - empno, ename, job, sal
no rows

create emp_mgr, emp_clerk, emp_others:

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

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

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

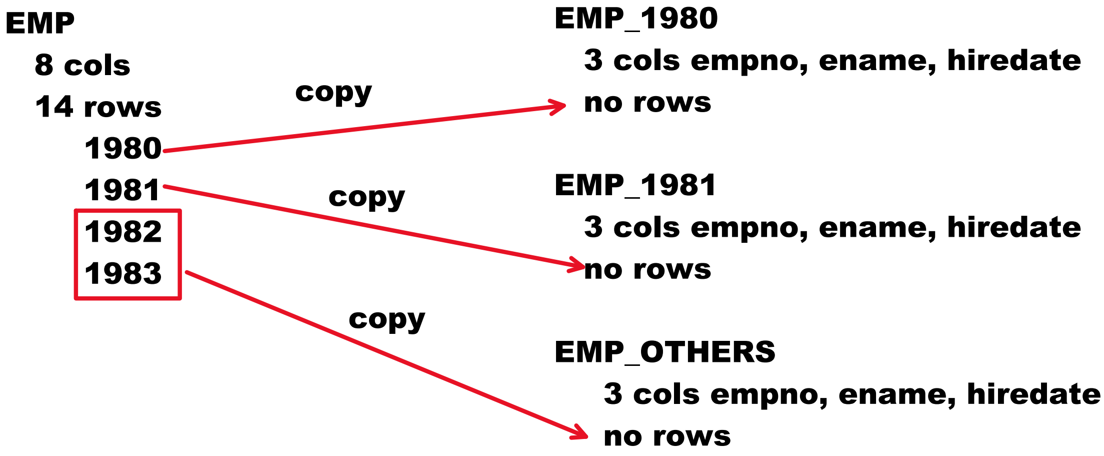
Copy managers records to emp_mgr
clerks emp_clerk
others emp_others:

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

```
SELECT * FROM emp;
```

Output:
14 rows created

Assignment:



EMP
8 cols
14 rows
1980
1981
1982
1983

copy

copy

copy

EMP_1980
3 cols empno, ename, hiredate
no rows

EMP_1981
3 cols empno, ename, hiredate
no rows

EMP_OTHERS
3 cols empno, ename, hiredate
no rows

```
WHEN to_char(hiredate,'yyyy')=1980 THEN
   INTO emp_1980 VALUES(empno, ename, hiredate)
```

MERGE

Monday, November 11, 2024 8:48 AM

Branch Office

s.cid = t.cid

Customer1 S

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

Head office

Customer2 T - REPLICA [dup copy]

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

- **MERGE command introduced in ORACLE 9i.**
- **MERGE = UPDATE + INSERT**
- **MERGE is a combination of UPDATE and INSERT commands.**
- **It can be also called as "UPSERT" command.**
- **It is used to apply one table changes to it's replica.**

Syntax:

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

Example:

Branch Office

Cusotmer1 S

CID	CNAME	CCITY
1001	A ABC	HYD BLR
1002	B	BLR
1003	C	DLH
1004	D	HYD

Customer2 T - REPLICA

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

1005	E	MUM

Apply customer1 table changes to it's replica:

```
MERGE INTO cusotmer2 T
USING cusotmer1 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);
```