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

Akhil (Admin)
Mobile: 9154156192 (Only WhatsApp)

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

Oracle [SQL/PL & SQL] @ 6:00 PM (IST) by Mr.Shiva Chaitanya
Day-1 https://youtu.be/YxWBKsT1CkA
Day-2 https://youtu.be/_hMQxtstz58
Day-3 https://youtu.be/Tdo4DAgZa-Y
Day-4 https://youtu.be/hF5GWPrncyU
Day-5 https://youtu.be/dD6VfK8d-pU
Day-6 https://youtu.be/haAhu8cGMQE
Day-7 https://youtu.be/TPIEsY4wvlw

ORACLE

Thursday, April 18, 2024 6:18 PM

Data Store => is a location where data is stored
Database => is a location where organization's business data stored
DBMS => is a software that is used to manage the DB
RDBMS => is a software that is used to manage DB in the form of tables
Metadata => data about the data

Data Store:

- **The location where data is stored**
- **Examples: BOOK, FILE, DATABASE**

GOAL: storing organization's business data permanently

BANK

Branches
Customers
Transactions
Employees
.
.

Before 1960s
business data in books

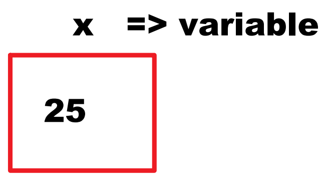
1960s => Files
1970s => DATABASES

In how many ways we can store the data in computer?

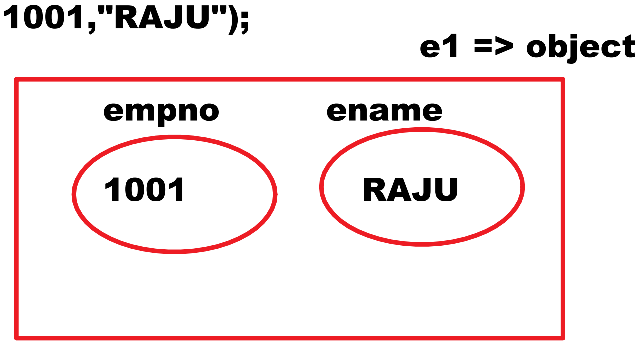
- **variable => temporary**
- **object => temporary**
- **File => permanent => drawbacks**
- **Database => permanent**

In Java:

int x=25;



Employee e1 = new Employee(1001,"RAJU");



File	**Database**
• it is suitable for 1 user only	• it is suitable for multiple users
• no security	• security
• suitable to store small amounts of data	• suitable to store large amounts of data

DATABASE: => data store [location]

BANK DB

Branches
Customers
Transactions

COLLEGE DB

Courses
Students
Fee

Customers
Transactions
Products
Employees
.
.

Students
Fee
Marks
Library
.
.

DB SERVER

searching for products
adding to wishlist
placing order
online payment

amazon DB

products
wishlists
customers
orders
payments

- **DATABASE is a kind of data store.**
- **DATABASE is a location where organization's business data stored permanently.**
- **DATABASE is a collection of interrelated data in an organized form.**

  interrelated =>
  BANK DB contains BANK related data only
  It does not contain COLLEGE related data.

  organized =>
  arranging in specific way

to visit websites	**Browser s/w**	**Google Chrome**
to watch the movies	**Media Player s/w**	**VLC**
to create the presentations	**Presentation s/w**	**MS powerpoint**

to maintain the database	**DBMS**	**ORACLE**

DBMS:

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

Before 1960s => BOOKS

In 1960s => FMS [File Management Software]

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

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

ORACLE company founder => LARRY ELLISON

1977 => Software Development Laboratories

1979 => renamed company => Relational Software Inc.
introduced ORACLE => first RDBMS

1983 => renamed company => ORACLE corp.

DBMS
H DBMS => nodes
N DBMS => nodes
R DBMS => tables

RDBMS:

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

BANK DB

BRANCHES Table

IFSC_CODE	CITY	STATE	COUNTRY

CUSTOMERS Table

CUSTID	CNAME	CCITY	MOBILE	AADHAR	PAN	IFSC_CODE

TRANSACTIONS table

T_ID	T_DATE_TIME	T_TYPE	ACNO	AMOUNT	CID

.
.

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

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

TABLE:
- **table is a collection of columns and rows.**
- **A Table can be also called as Relation / Entity**

Example:

EMPLOYEE → **Table / Relation / Entity**

EMPID	ENAME	SAL
1001	kiran	8000
1002	sai	6000

Column / Field / Attribute / Property

Row / Record / Tuple / Entity Instance

Column:
Vertical representation of data is called "Column"

Row:
Horizontal representation of data is called "Row".

Metadata:

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

Examples:
Field names => sid, sname, fee
Table name => student

Example:
STUDENT

SID	SNAME	FEE
1001	Kiran	6000

BANK DB

BRANCHES

IFSC_CODE	CITY	STATE	COUNTRY

CUSTOMERS

CUSTID	CNAME	CCITY	MOBILE	AADHAR	PAN	IFSC_CODE

TRANSACTIONS

T_ID	T_DATE_TIME	T_TYPE	ACNO	AMOUNT	CID

-
-

ORACLE

Friday, April 19, 2024 6:41 PM

ORACLE:

- **is a Relational DataBase Management software.**
- **it is used to create and maintain the database in the form of tables.**
- **Using ORACLE, we can store, manipulate and retrieve the data of database.**

  manipulate => insert / update / delete

  emp joined => insert
  emp sal increased => update [modify]
  emp resigned => delete

  retrieve => opening existing data

  checking balance
  transactions statement
  searching for products

- **ORACLE software 2nd version released in 1979.**
  they didn't 1st version to market.
- **Latest version is:**

For Windows OS	**ORACLE 21C**
For LINUX OS	**ORACLE 23C**

Before 1960s => Books

In 1960s => FMS

In 1970s => HDBMS
NDBMS

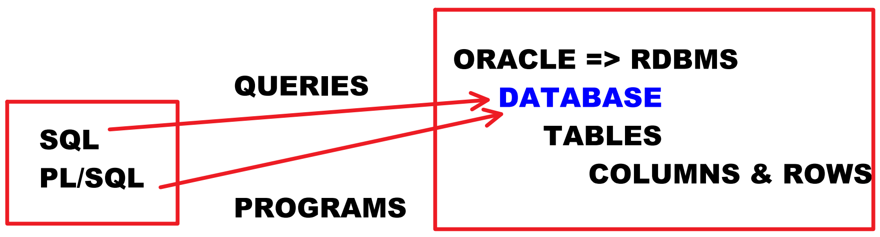
In 1976 => RDBMS concept => E.F.Codd

ORACLE Company Founder => Larry Ellison

In 1977 => larry ellison established => Software Development Laboratories
In 1979 => company name renamed => Relational Software Inc.

introduced ORACLE 2nd version

In 1983 => company name renamed => ORACLE CORP.



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

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

SQL:

- **SQL => Structured Query Language**
- **It is a Query Language**
- **It is used to write the queries**
- **Query => request / instruction / command**
- **Query is a request that is sent to DB SERVER.**
- **We write queries in SQL to communicate with ORACLE DB.**

C, Java, C#:

Software
Programs
Instructions

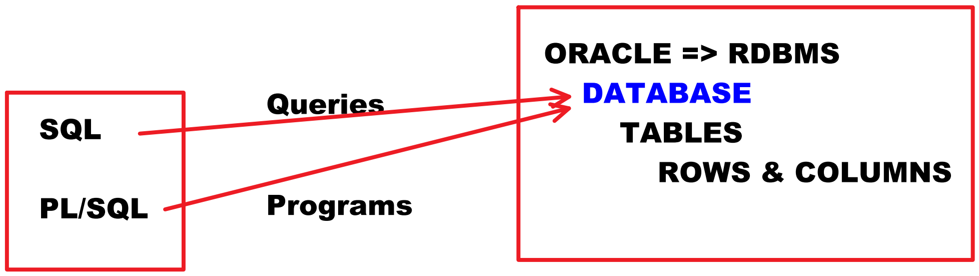
PL/SQL:

- **PL => Procedural Language**
- **SQL => Structured Query Language**
- **It is a Programming Language.**
- **in this, we develop the programs to communicate with ORACLE DB**

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

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

Saturday, April 20, 2024 6:16 PM



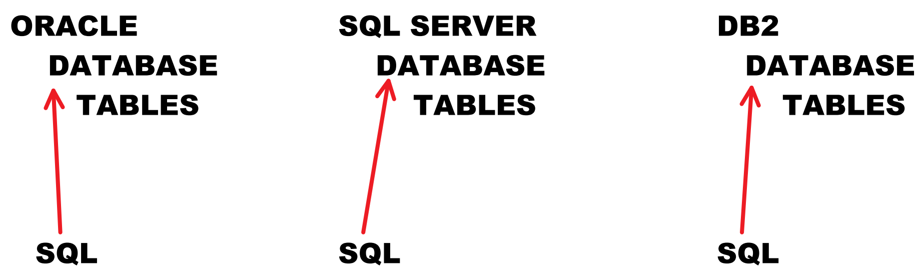
SQL:

- **SQL => Structured Query Language.**
- **It is a query language.**
- **it is used to write the queries.**
- **we write queries to communicate with ORACLE DB.**
- **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 a Unified Language. It is common language to work with many Relational databases.**

In C:
Function:
is a set of statements

In Java:
Method:
is a set of statements

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



- **SQL provides built-in functions to make our actions easier.**
- **SQL provides operators to perform operations.**
- **SQL provides JOINS concept to retrieve data from multiple tables.**

SQL provides JOINS concept to retrieve data from multiple tables.

STUDENT

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

COURSE

CID	CNAME
10	JAVA
20	PYTHON
30	C#

JOINS

SNAME	CNAME
student	course

- **SQL provides readymade commands.**

SQL sub languages:

SQL provides 5 sub languages. They are:

- **DDL**
- **DRL / DQL**
- **DML**
- **TCL**
- **DCL / ACL**

Every Sub Language provides Commands. These are called "SQL commands".

DDL: • **Data Definition Language** • **Data Definition => metadata** • **it deals with metadata**	**create** **alter** **drop** **flashback [oracle 10g]** **purge [oracle 10g]** **truncate** **rename**
DRL / DQL:	**select**

• **Data Retrieval Language** • **Data Query Language** • **retrieve => opening existing data** • **it deals with data retrievals**	
DML • **Data Manipulation Language** • **manipulation => 3 actions** **insert/update/delete**	**insert** **update** **delete** **insert all [oracle 9i]** **merge [oracle 9i]**
TCL: • **Transaction Control Language** • **It deals with transactions**	**commit** **rollback** **savepoint**
DCL / ACL: • **Data Control Language** • **Accessing Control Language** • **It deals with data accessibility**	**grant** **revoke**

DDL:
create
alter

drop
flashback
purge

truncate

rename

create:
create command is used to create oracle db objects like tables, views .... etc.

ORACLE DB Objects
- **TABLES**
- **VIEWS**
- **INDEXES**
- **SEQUENCES**

create oracle db objects like
tables, views .... etc.

VIEWS
INDEXES
SEQUENCES
SYNONYMS
M.VIEWS
PROCEDURES
FUNCTIONS
PACKAGES
TRIGGERS

TABLE EMPLOYEE

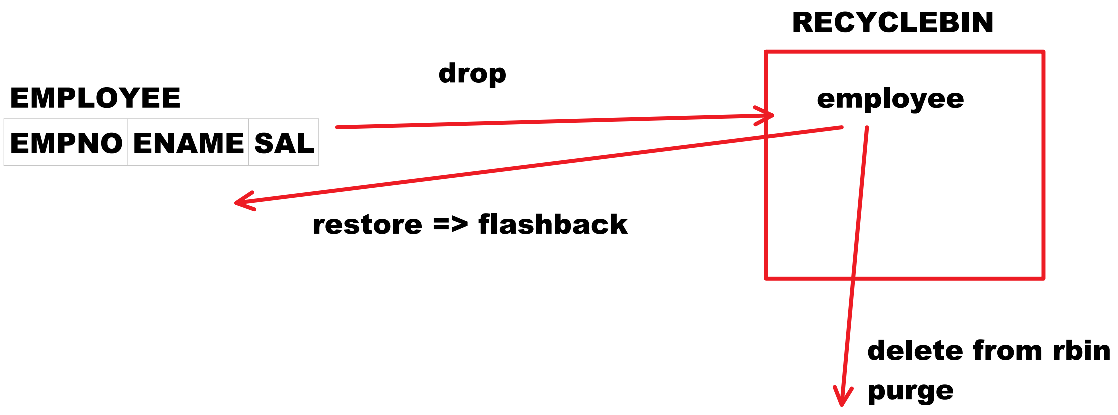
EMPNO	ENAME	SAL

alter:

- alter => change
- used to change structure of the table.
- using this, we can add the columns. rename the columns or drop the columns.

drop
flashback [oracle 10g]
purge [oracle 10g]

In oracle 10g, a new feature added i.e. RECYCLEBIN



truncate:

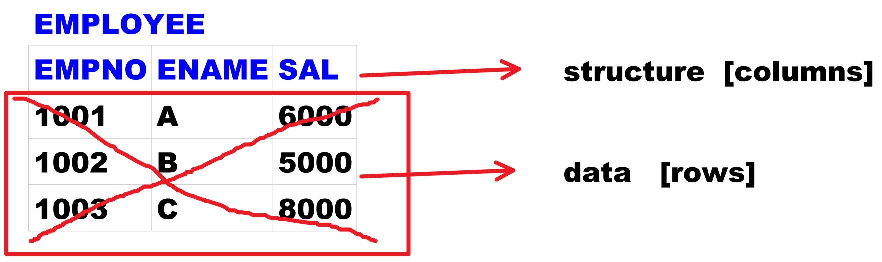


table = structure + data

rename:
to change table name we use it

SYLLABUS

Monday, April 22, 2024 6:14 PM

ORACLE

TABLES

SQL

1	SQL Commands	DDL, DRL, DML, TCL, DCL
2	Built-In Functions	
3	CLAUSES	GROUP BY, HAVING
4	JOINS	
5	SUB QUERIES	
6	CONSTRAINTS	
7	VIEWS	
8	INDEXES	
9	SEQUENCES	
10	SYNONYMS	
11	MATERIALIZED VIEWS	

PL/SQL

PL/SQL Basics	data types, declare, assign, print, read
Control Structures	
CURSORS	
COLLECTIONS	
EXCEPTION HANDLING	
STORED PROCEDURES	
STORED FUNCTIONS	
PACKAGES	

TRIGGERS	
WORKING WITH LOBs	
DYNAMIC SQL	

SQL

Monday, April 22, 2024 6:26 PM

DDL:

CREATE:

used to create the tables.

Syntax:

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

[ ]	Optional
< >	Any

For WINDOWS OS, Latest version is: ORACLE 21C
For LINUX OS, Latest version is: ORACLE 23C

Till ORACLE 21C, we can create max of 1000 columns.
In ORACLE 23C, we can create max of 4096 columns.

Data Types in SQL:

Data Type tells,
- how much memory has to be allocated
- which type of data should be accepted in column

ORACLE SQL provides following data types:

Character Related
 	
Examples:
'RAJU'
'MANAGER'
'B.Tech'
	Char(n)
	Varchar2(n)
	LONG
	CLOB
	
	nChar(n)
	nVarchar2(n)
	nCLOB

Integer Related Examples: 1234 21	NUMBER(p) Integer Int
Floating Point Related Examples: 1600.80 8000.00 67.89	NUMBER(p,s) Float binary_float binary_double
Date and time Related Examples: 25-DEC-23 22-APR-24 6:54:0.0 PM	Date Timestamp [ORACLE 9i]

Binary Related **Examples:** **images, audios, videos, documents ... etc**	**BFILE** **BLOB**

Character Related data types:

Char(n):

- **n => max no of chars**
- **it is used to hold string values.**
- **It is Fixed Length Data Type.**
- **max size: 2000 Bytes [2000 chars]**
- **default size: 1**
- **to hold fixed length chars use "CHAR"**

Varchar2(n):

- **n => max no of chars**
- **it is used to hold string values.**
- **It is Variable Length Data Type.**
- **max size: 4000 Bytes [4000 chars]**
- **default size: no default size**
- **to hold variable length chars use "VARCHAR2".**

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

```
VEHICLE_NUM   CHAR(10)                ENAME VARCHAR2(10)
-----------------------               ------------
TS09AA1234                            kiran
                                      naresh
                                      sai


PAN_NUMBER   CHAR(10)               job        VARCHAR2(10)
-----------------------             --------
ABCDE1234F                          manager
                                    clerk


GENDER   CHAR(1)                  mail_id   VARCHAR2(30)
-----------                       ------------
M                                 raju1234@gmail.com
F                                 sai@gmail.com
```

VARCHAR2 data type can hold max of 4000 chars only.
To hold more than 4000 chars we can use LONG or CLOB.

LONG:

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

- max size: 2GB

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

Examples:

feedback CLOB

complaints CLOB

product_features CLOB

Character related data types:

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

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

In C:
```
    char ch;    // 1 Byte => ASCII => english lang only
```

In Java:
```
    char ch;    // 2 Bytes  => UNI => english lang + other lang
```

ASCII:
- ASCII => American Standard Code for Information Interchange
- is a coding system
- 256 chars are coded.
- ranges from 0 to 255.
- 255 => 1111 1111 => 8 bits => 1 Byte
- English lang chars + digits + special chars

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

UNI:
- UNI => UNIversal
- is a coding system
- extension of ASCII
- 65536 chars are coded.
- ranges from 0 to 65535
- 65535 => 1111 1111 1111 1111 => 16 bits [2 Bytes]
- English lang chars +digits + special chars + other lang chars

Integer related data types:

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

Examples:

```
empno NUMBER(4)          -9999 TO 9999
-----------
1234
1235
```

1236
5678
67
9999
10000 => ERROR

max marks: 100

```
maths NUMBER(3)              -999 TO 999
---------
78
100
678
999
9
1000  => ERROR
```

Aadhar_number NUMBER(12)

Mobile_number NUMBER(10)

Credit_Card_number NUMBER(16)

Note:
What are the differences between Number(38), integer and int?

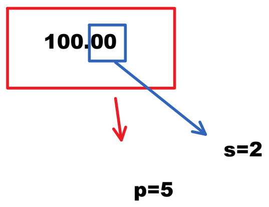
all are same
integer and int are alias names of NUMBER(38)

Integer = int = NUMBER(38)

Floating point related data types:

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

Example:



```
avrg  NUMBER(5,2)        -999.99 TO 999.99
----------
67.89
100.00
999.99
1000    => ERROR
123.456789  => 123.46
```

123.453789 => 123.45

max sal:
00000.00

salary NUMBER(8,2)   -999999.99 TO 999999.99

height   NUMBER(2,1)   -9.9 TO 9.9

5.3
5.0
5.8
5.9

Date & Time related data types:

Date:

- it is used to hold date values.
- it can hold date, month, year, hours, minutes and seconds.
- default date format is: 'DD-MON-YY'
- Example: 23-APR-24
- date also contains time value. But, by default it will not be displayed.
- default time: 12:00:00 AM [mid night time]
- it is fixed length data type
- size: 7 Bytes

Examples:

Transaction_date DATE
date_of_birth DATE
ordered_date DATE

Timestamp:

- Introduced in ORACLE 9i
- Used to hold date and time values.
- it can hold date, month, year, hours, minutes, seconds and fractional seconds.
- it is fixed length data type.
- size: 11 Bytes
- default time: 12:00:0.0 AM [mid night time]
- format: DD-MON-YY HH:MI:SS.FF AM

Examples:

Transaction_date_time TIMESTAMP

ordered_date_time TIMESTAMP

delivered_date_time TIMESTAMP

login_date_time TIMESTAMP

Differences b/w DATE and TIMESTAMP:

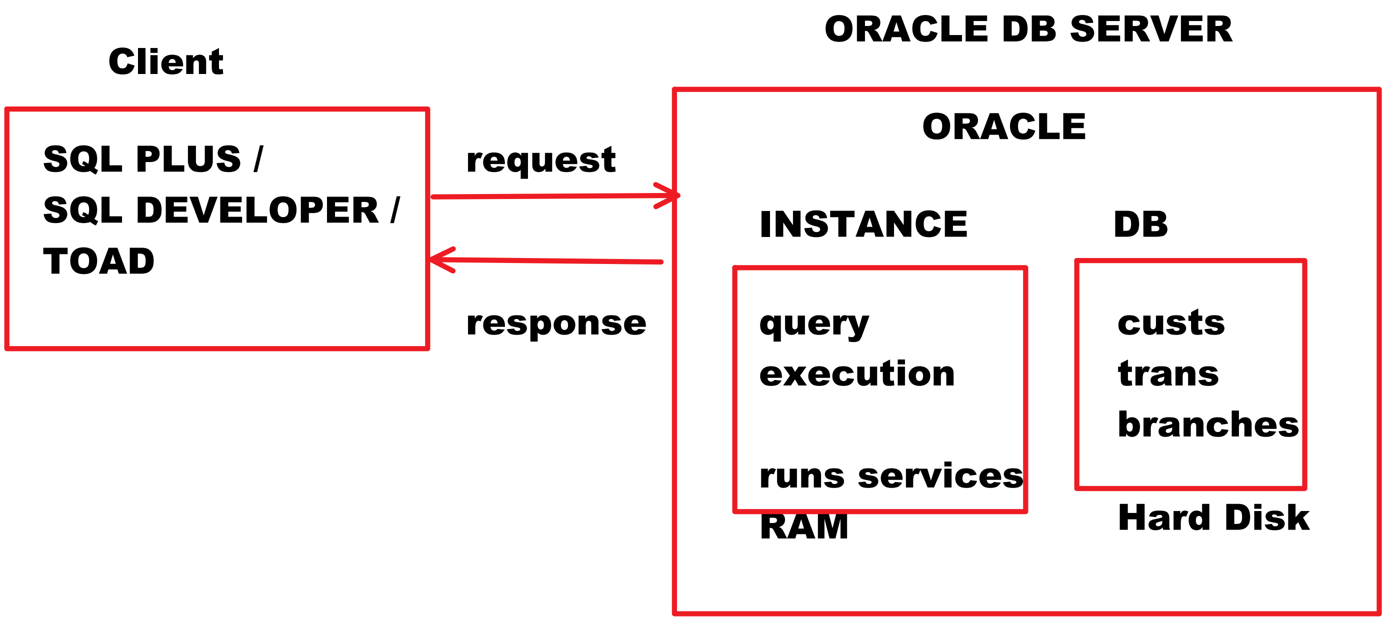
DATE	TIMESTAMP
• it cannot hold fractional seconds	• it can hold fractional seconds
• size: 7 Bytes	• size: 11 Bytes
• it does not display time by default	• it displays time by default
• it is used to hold date values	• it is used to hold date and time
Example: transaction_date DATE	Example: trans_date_time TIMESTAMP

Monday, April 22, 2024 7:07 PM

Fixed length **Variable Length**

T1

Fixed length	F1 CHAR(10)	F2 VARCHAR2(10)	Variable Length
10	raju6spaces	raju	4
10	naresh4spaces	naresh	6
10	sai7spaces	sai	3



Note:
When we install ORACLE software, along with it also installs SQL PLUS.

Opening SQL PLUS:

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

Creating User:

Syntax to create the User:

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

username: c##batch6pm
password: nareshit

common user	**c##raju**
local user	**raju**

Note:
DBA creates the user

Login as DBA:
username: system
password: naresh

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

        Output:
        user created.

SQL> GRANT connect, resource, unlimited tablespace
     TO c##batch6pm;

        Output:
        Grant Succeeded
```

connect	• **is a privilege [permission]** • **is a permission for log in**
resource	• **is a privilege [permission]** • **is a permission to create the tables**

unlimited tablespace	• is a privilege [permission] • is a permission to insert the records

to see current username:

```
SQL> show user
```

to clear screen:

```
SQL> cl scr
```

Note:
CL[EAR] SCR[EEN]

to login from sql command prompt:

Syntax:

```
conn[ect] <username>/<password>
```

Example:

```
SQL> conn c##batch6pm/nareshit
```

Modifying user's password:

Syntax:

```
ALTER USER <user name>
ALTER USER <user_name>
IDENTIFIED BY <new_password>;
```

Example:

```
ALTER USER c##batch6pm
IDENTIFIED BY naresh;
```

Changing DBA password:

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

SQL> ALTER USER system
        IDENTIFIED BY nareshit;
```

Dropping User:

Syntax:

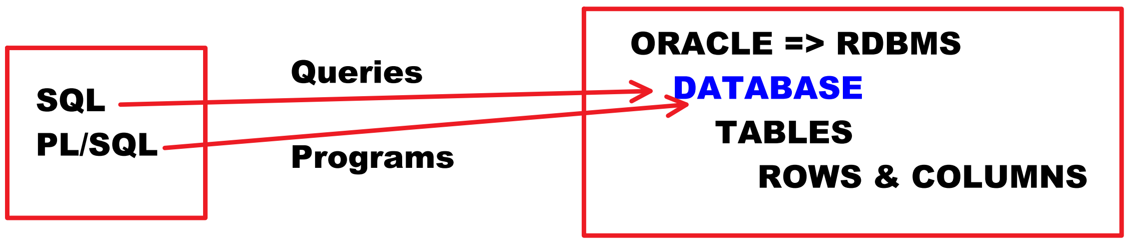
```
DROP USER <user_name> CASCADE;
```

Example:

```
DROP USER c##abcd CASCADE;
```

Creating user:

username: c##abcd
password: abcd



SQL Commands:

SQL provides 5 Sub Languages:

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

CREATE:

- **it is DDL command.**
- **it is used to create the tables.**

Syntax to create the table:

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

INSERT:

- **it is DML command.**
- **it is used to insert the records.**

Syntax:

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

Note:
SQL is not case sensitive language
select = SELECT = SELecT

Examples on creating tables and inserting records:

Example-1:

MAX AVRG:
100.00

STUDENT

SID	SNAME	AVRG
1001	AA	67.89
1002	ABC	56.23

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

creating table:

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

inserting records:

1001	AA	67.89
1002	ABC	56.23

```
INSERT INTO student VALUES(1001,'AA',67.89);  --inserts in INSTANCE [RAM]
Output:
1 row created.

INSERT INTO student VALUES(1002,'ABC',56.23);   --inserts in INSTANCE [RAM]
Output:
1 row created.

COMMIT;      --data will be moved from INSTANCE to DB [RAM to HARD DISK]
```

to open and see table data:

```
SELECT * FROM student;
Output:
```

SID	SNAME	AVRG
1001	AA	67.89
1002	ABC	56.23

Inserting records using parameters:

- **parameter concept is used to read the value at run time.**

Syntax:

&<text>

Example:

```
INSERT INTO student VALUES(&sid,'&sname',&avrg);
Output:
enter value for sid: 1003
enter value for sname: XYZ
enter value for avrg: 78.54
INSERT INTO student VALUES(&sid,'&sname',&avrg)
INSERT INTO student VALUES(1003,'XYZ',78.54)
1 row created

/
Output:
enter value for sid: 1004
enter value for sname: A
enter value for avrg: 55.66

/
Output:
enter value for sid: 1005
enter value for sname: B
enter value for avrg: 44.45
```

Note:
/ is used to run recent command in memory
it means, / runs above query [recent query]

/	R[UN]

string must be enclosed in single quotes

```
INSERT INTO student VALUES(&sid,'&sname',&avrg);
Output:
.....
...enter value for sname: RAJU
```

INSERT INTO student VALUES(&sid,&sname,&avrg);
Output:
.....
...enter value for sname: 'RAJU'

Inserting limited column values:

STUDENT

SID	SNAME	AVRG
2001	AA	

INSERT INTO student VALUES(2001,'AA');
Output:
ERROR: not enough values

INSERT INTO student(sid,sname) VALUES(2001,'AA');

STUDENT

SID	SNAME	AVRG
2002		52.82

INSERT INTO student(sid,avrg) VALUES(2002,52.82);

Inserting Limited Column Values by changing order of columns:

STUDENT

SID	SNAME	AVRG
2003	XYZ	

INSERT INTO student(sname,sid) VALUES('XYZ',2003);

Example-2:

EMPLOYEE

EMPNO	ENAME	STATE	SAL	DOJ
1234	ABC	TS	12000	25-DEC-23
1235	AB	AP	15000	17-AUG-21

100000.00

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

creating table:

EMPLOYEE

EMPNO	ENAME	STATE	SAL	DOJ

```
CREATE TABLE employee
(
empno NUMBER(4),
ename VARCHAR2(10),
state CHAR(2),
sal NUMBER(8,2),
doj DATE
);
Output:
table created.
```

1234	ABC	TS	12000	25-DEC-23
1235	AB	AP	15000	17-AUG-21

```
INSERT INTO employee
VALUES(1234,'ABC','TS',12000,'25-DEC-2023');
                              string
```

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

```
INSERT INTO employee
VALUES(1235,'AB','AP',15000,'17-AUG-2021');
```

string

```
INSERT INTO employee
VALUES(1235,'AB','AP',15000,'17-AUG-2021');

COMMIT;
```

implicit conversion

Note:

- **implicit conversion degrades the performance.**
- **to improve performance, do explicit conversion.**
- **for explicit conversion we use Built-In Functions.**

1236	XYZ	MH	10000	27-FEB-22

```
DOJ
--------
27-FEB-22 DATE
```

```
INSERT INTO employee
VALUES(1236,'XYZ','MH',10000,to_date('27-FEB-2022'));
```

string

to_date()
explicit conversion

1237	Kiran	TS	8000	25-SEP-2023

```
INSERT INTO employee
VALUES(1237,'Kiran','TS',8000,to_date('25-SEP-2023'));
```

string

to_date()
explicit conversion

```
DOJ
----------
25-SEP-23   date
```

Inserting emp record with today's date:

sysdate:

- **it is a built-in function.**
- **it is used to get current system date.**

1238	Raju	AP	13000	sysdate

```
INSERT INTO employee
VALUES(1238,'Raju','AP',13000,sysdate);
```

Example-3:

EMP1

EMPNO	ENAME	LOGIN_DATE_TIME
1001	A	25-APR-24 10:30:0.0 AM
1002	B	25-APR-24 2:30:0.0 PM

creating table:

```
CREATE TABLE emp1
(
empno NUMBER(4),
ename VARCHAR2(10),
login_date_time TIMESTAMP
);
```

inserting records:

1001	A	25-APR-24 10:30:0.0 AM
1002	B	25-APR-24 2:30:0.0 PM

```
INSERT INTO emp1 VALUES(1001,'A','25-APR-2024 10:30 AM');
output:
error

INSERT INTO emp1 VALUES(1001,'A','25-APR-2024 10:30:0.0 AM');
output:
1 row created.
```

string
\

output:
1 row created.

string

implicit conversion

login_date_time

25-APR-24 10:30:0.0 AM **timestamp**

1002	**B**	**25-APR-24 2:30:0.0 PM**

```
INSERT INTO emp1
VALUES(1002,'B',to_timestamp('25-APR-2024 2:30:0.0 PM'));
```

string

to_timestamp()
explicit conversion

login_date_time

25-APR-2024 2:30:0.0 PM **timestamp**

inserting record with current system date and time:

1003	**C**	**systimestamp**

systimestamp:
- **it is a built-in function**
- **it is used to get current system date and time**

```
INSERT INTO emp1
VALUES(1003,'C',systimestamp);
```

Note:
To see table strcture:

DESC[RIBE]:
it is used to see table structure

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

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

To see all tables list which are created by a user:

user_tables:

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

```
DESC user_tables

SELECT table_name FROM user_tables;
```

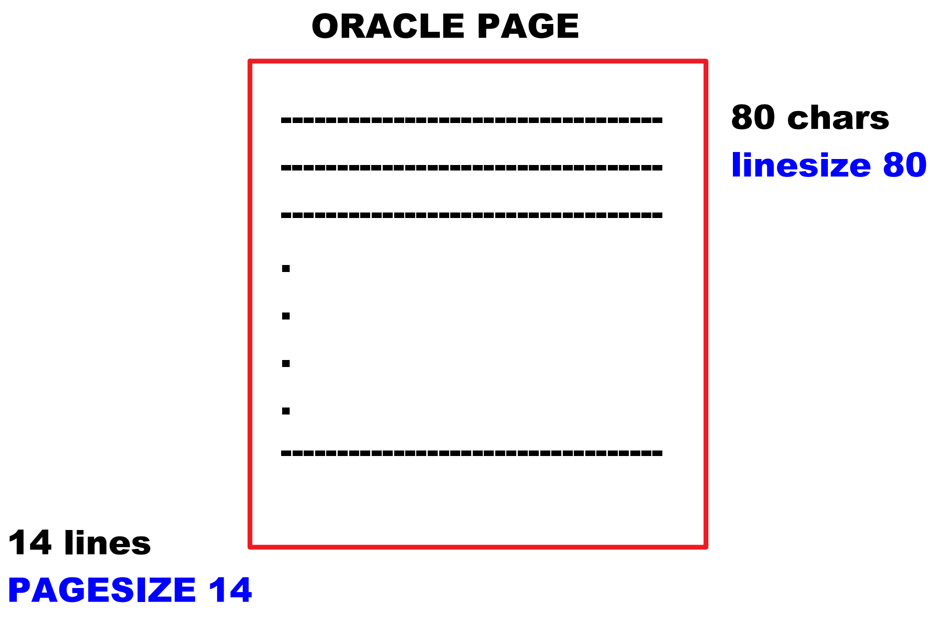
default PAGESIZE is 14
default LINESIZE is 80

to see all parameters list:

```
SQL> SHOW ALL
```

Output:

```
LINESIZE     80
PAGESIZE     14
```



To set page size:

```
SQL> SET PAGES 200
```

To set line size:

```
SQL> SET LINES 200
```

To set page size and line size:

```
SQL> SET PAGES 200 LINES 200
```

Column Alias

Monday, April 29, 2024 6:23 PM

Column Alias:

- **Column alias => another name or alternative name for column**

- **To change column headings in output we use COLUMN ALIAS.**

- **to give column alias we use AS keyword.**
  Using AS keyword is optional.

- **to give column alias in multiple words or to maintain the case specify column alias in double quotes.**

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
```

DRL

Friday, April 26, 2024 7:34 PM

DRL / DQL:

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

- **Retrieve => opening existing data**
- **Query => is a request that is sent to DB SERVER.**

- **It deals with data retrievals.**

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

SELECT:

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

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

Syntax of SELECT command:

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

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

CLAUSE => is a part of query

every CLAUSE has specific purpose
every QUERY is made up of with CLAUSES.

All columns, All rows:

Display all emp table columns and rows:

```
SELECT * FROM emp;
```

*	All Columns

note:
SELECT * FROM emp;
above query will be rewritten by oracle as following:

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

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

- **All columns, specific rows:**

  Display the emp records whose salary is 3000:

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

- **Specific columns, All rows:**

  Display all emp names and salaries:

```
SELECT ename, sal
```

```
FROM emp;
```

Specific columns, specific rows:

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

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

All Columns	**SELECT ***
Specific Columns	**SELECT ename,sal**
All Rows	**Don't write WHERE condition**
Specific Rows	**Write WHERE condition**

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 / Comparison	**> < >= <= = (equals) != / <> / ^= (not equals)**
Logical	**AND OR NOT**

Special / Comparison	IN BETWEEN AND LIKE IS NULL Exists Any All	NOT IN NOT BETWEEN AND NOT LIKE IS NOT NULL
Set	UNION UNION ALL INTERSECT MINUS	
Concatenation	||	

Arithmetic operators:
Arithmetic operators are used to perform Arithmetic operations like addition , subtraction ... etc.

ORACLE SQL provides following Arithmetic Operators:

+	Addition
-	Subtraction
*	Multiplication
/	Divison

In C/Java:
5/2 = 2
int/int = int

5%2 = 1

In ORACLE SQL:
5/2 = 2.5
NUMBER/NUMBER = NUMBER

MOD(5,2) = 1

Examples on Arithmetic Operators:

Calculate Annual salary of all emps:

```
SELECT ename, sal, sal*12
FROM emp;
Output:
ENAME        SAL        SAL*12
------------------------------------------------------
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
------------------------------------------------------
SMITH         800        9600
ALLEN        1600       19200
```

Example:

Calculate TA, HRA, TAX and GROSS salary of all emps:
10% on sal => TA
20% on sal => HRA
5% on sal => TAX
GROSS = sal + ta + hra - tax

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

Calculate experience of all emps:

```
select ename, sal,
trunc((sysdate-hiredate)/365) as experience
from emp;
```

display the emp records who are ahaving more than 42years experience:

```
SELECT ename, hiredate
FROM emp
WHERE trunc((sysdate-hiredate)/365)>42;
```

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

```
SELECT ename, sal, sal*12 as annual_sal
FROM emp
WHERE sal*12>35000;
```

Assignment:

STUDENT

SID	SNAME	M1	M2	M3
1001	A	70	90	80
1002	B	44	78	39

calculate total and average marks of all students.

PLAYER

PID	PNAME	runs	balls
1001	A	20	10
1002	B	30	20

calculate strike rate of each player

strike rate = runs*100/balls

Relational Operators / Comparison Operators:

- **Relational operator is used to compare column value with 1 value.**
- **ORACLE SQL provides following Relational Operators:**

<
>
>=
<=

sal>3000	valid
sal>3000,4000	invalid
sal=3000	valid

sal>3000,4000	invalid
sal=3000	valid
sal=3000,4000	invalid

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

Examples on relational operators:

Display the emp records whose salary is more than 2500:

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

Display the emp records whose salary is 3000 or more:

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

Display the emp records whose salary is less than 1000:

```
SELECT ename, sal
FROM emp
WHERE sal<1000;
```

Display the emp records whose salary is more than 3000:

emp

empno	ename	sal
1001	A	5000
1002	B	3000

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

1001	A	5000
1002	B	3000
1003	C	8000
1004	D	1000

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

Execution Order:
FROM
WHERE
SELECT

FROM emp:
entire emp table will be selected.
FROM clause selects entire table.

empno	ename	sal
1001	A	5000
1002	B	3000
1003	C	8000
1004	D	1000

WHERE sal>3000:
WHERE condition will be applied on every row.
WHERE clause filters the rows.

empno	ename	sal
1001	A	5000
1002	B	3000
1003	C	8000
1004	D	1000

```
WHERE sal>3000
----------------------------------
5000>3000 T
3000>3000 F
8000>3000 T
1000>3000 F
```

empno	ename	sal
1001	A	5000
1003	C	8000

SELECT ename, sal :

- **it selects ename and sal columns.**
- **SELECT clause selects specified columns.**

ename	sal
A	5000
C	8000

Note:
CALENDAR order is ASCENDING ORDER [small to big]

2023 calendar

1-JAN-2023 min date
2-JAN-2023
3-JAN-2023
.
.
31-DEC-2023 max date

after 2023:

31-DEC-2023
$\boxed{\begin{array}{l}\textbf{1-JAN-2024}\\ \textbf{10-FEB-2024}\end{array}}$ **hiredate>'31-DEC-2023'**

before 2023:

$\boxed{\begin{array}{l}\textbf{17-AUG-2022}\\ \textbf{31-DEC-2022}\end{array}}$ **hiredate < '1-JAN-2023'**
1-JAN-2023

Display the emp records who joined after 1981:

31-DEC-1981

```
1-JAN-1982
2-JAN-1982
.
.
```

hiredate > '31-DEC-1981'

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

Display the emp records who joined before 1981:

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

Display managers records:

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

empno	ename	job
1001	A	CLERK
1002	B	MANAGER

```
WHERE job='manager'
-----------------------------------
CLERK=manager        F
MANAGER=manager  F
```

when all conditions are FALSE, we get output as "no rows selected".

Note:
SQL is not case sensitive language. But,
string comparison is case sensitive.

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

Display the emp record whose empno is 7521:

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

Display the emp records who are working in deptno 20:

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

Display all emp records except managers:

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

Logical operators:

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

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

AND, OR are used to separate multiple relational conditions.

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

Truth Table:

c1 => condition1
c2 => condition2

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

Examples on AND, OR:

Display all managers and clerks records:

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

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

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

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

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

Display the emp records whose salary is 2000 or more and 3000 or less [whose salary between 2000 and 3000]

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

Display the managers records who are earning more than 2500:

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

Display the managers records who joined after april 1981:

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

Display the emp records who joined in 1982:

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

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

Display SMITH, BLAKE and SCOTT records:

```
SELECT ename,sal
FROM emp
WHERE ename='SMITH' OR ename='BLAKE' OR ename='SCOTT';
```

Online Shopping

FILTER

```
WHERE cname='DELL'

WHERE cname='DELL' OR cname='MICROSOFT'

WHERE cname='DELL' AND price>=50000 AND price<=70000

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

NOT:

It is used to perform logical NOT operations.

Truth Table:

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

Display all emp records except managers:

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

(or)

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

Example:

STUDENT1

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

```
CREATE TABLE student1
(
sid NUMBER(4),
sname VARCHAR2(10),
m1 NUMBER(3),
m2 NUMBER(3),
m3 NUMBER(3)
);

INSERT INTO student1 VALUES(1001,'A',70,90,80);
INSERT INTO student1 VALUES(1002,'B',80,30,60);
```

```
COMMIT;
```

Display passed students records:

```
SELECT *
FROM student1
WHERE m1>=40 AND m2>=40 AND m3>=40;
```

Display failed students records:

```
SELECT *
FROM student1
WHERE m1<40 OR m2<40 OR m3<40;
```

Special Operators:

- **Special operators can be also called as Comparison Operators.**

IN:

Syntax:

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

- **it is used to compare column value with a list of values.**

- **it avoids of writing multi equality conditions using OR.**

examples on IN operator:

Display the emp records whose salary is 1250 or 3000:

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

(or)

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

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

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

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

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

Display all managers and clerks records:

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

Display all emp records except managers and clerks:

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

If job value NOT IN list then condition is TRUE

Display all emp records except deptno 10 and 30:

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

BETWEEN .. AND:

Syntax:

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

- **i8t is used to compare column value with range of values.**

Examples on BETWEEN .. AND:

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

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

```
WHERE sal BETWEEN 2000 AND 3000;
```

Display the emp records who joined in 1982:

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

Display the emp records whose salary is less than 2000 or more than 3000 [whose salary not between 2000 and 3000]:

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

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

what is the output of above query?

A. displays emp records whose sal b/w 2000 and 3000
B. no rows selected
C. Error
D. None of the above

Answer: B

LIKE:

Syntax:

Syntax:

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

- **It is used to compare column value with text pattern**
- **TO specify text pattern ORACLE SQL provides 2 wildcard characters:**

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

Examples on LIKE:

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

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

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

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

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

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

Display the emp records whose names are having M letter:

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

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

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

Display the emp records whose name has 4 letters:

```
SELECT ename, sal
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 getting 3 digits salary:

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

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 for null comparison.

Syntax:

```
<column> IS null
```

Examples:

Display the emp records who are not getting commission:

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

Display the emp records who are getting commission:

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

Concatenation Operator:

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

```
EMP1
FNAME  LNAME
------------------------
RAJ       KUMAR
SAI       TEJA
```

```
SELECT fname || ' ' || lname FROM emp1;
Output:
RAJ KUMAR
```

Example:
Display output as following:
SMITH works as CLERK
ALLEN works as SALESMAN
BLAKE works as MANAGER

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

Disaply output as following:
SMITH joined on 17-DEC-80
ALLEN joined on 25-FEB-81

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

NULL

Wednesday, May 1, 2024 7:26 PM

NULL:

- **NULL means empty / blank**

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

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

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

  Example:

  ```
  SELECT 100+200 FROM dual;          --300
  SELECT 100+200+null FROM dual;   --null
  ```

- **For NULL comaprison we cannot use = [equals]. we must use "IS NULL"**

We can insert NULL in 2 ways:

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

Example:

EMPLOYEE1

EMPNO	ENAME	SAL

```
CREATE TABLE employee1
(
empno NUMBER(4),
ename VARCHAR2(10),
sal NUMBER(8,2)
);
```

1001	A	7000

```
INSERT INTO employee1 VALUES(1001,'A',7000);
```

1002	B	

direct way: using NULL keyword:

```
INSERT INTO employee1 VALUES(1002,'B',null);
```

1003		8000

direct way: using NULL keyword:

```
INSERT INTO employee1 VALUES(1003,null,8000);
```

1004	D	

indirect way: insert limited column values

```
INSERT INTO employee1(empno,ename) VALUES(1004,'D');
```

STUDENT

SID	SNAME	M1 NUMBER(3)
1001	A	70
1002	B	80
1003	C	0
1004	D	55
1005	E	$\boxed{}$

$\longrightarrow$ null
we are unable to insert
ABSENT

EMPLOYEE

EMPNo	ENAME	SAL
1001	A	6000
1002	B	8000
1003	C	$\boxed{}$
1005	D	7000

$\longrightarrow$ null
salary is unknown

UPDATE

Thursday, May 2, 2024 6:53 PM

UPDATE:

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

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

Syntax:

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

Examples on UPDATE:

modifying single value of single record:

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

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

Set job as MANAGER, sal as 6000 to an employee whose empno is 7369:

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

modifying a group of records:

Increase 20% on sal to all managers:

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

modifyng all records:

Increase 1000 rupees salary to all emps:

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

Transfer all deptno 10 emps to deptno 20:

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

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

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

set comm as null to the emp whose empno is 7499:

```
UPDATE emp
SET comm=null
WHERE empno=7499;
```

Note:
For null comparison we cannot use =
For null assignment we use =

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

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

Increase 20% on salary to the emps who are having more than 42years experience:

```
UPDATE emp
SET sal=sal+sal*0.2
WHERE TRUNC((sysdate-hiredate)/365)>42;
```

Example:

EMPLOYEE5

EMPNO	ENAME	SAL	TA	HRA	TAX	GROSS
1001	A	8000				
1002	B	6000				

Calculate TA, HRA, TAX and GROSS salary:

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

```
CREATE TABLE employee5
(
empno NUMBER(4),
ename VARCHAR2(10),
sal NUMBER(8,2),
TA NUMBER(8,2),
HRA NUMBER(8,2),
TAX NUMBER(8,2),
GROSS NUMBER(8,2)
);


INSERT INTO employee5(empno,ename,sal)
VALUES(1001,'A',8000);

INSERT INTO employee5(empno,ename,sal)
VALUES(1002,'B',6000);

COMMIT;
```

calculte TA, HRA, TAX and GROSS:

```
    UPDATE employee5
    SET TA=SAL*0.1, HRA=SAL*0.2, TAX=SAL*0.05;

    UPDATE employee5
    SET gross = sal+ta+hra-tax;

    COMMIT;
```

ASSIGNMENT:

STUDENT

SID	SNAME	M1	M2	M3	TOTAL	AVRG
1001	A	70	90	50		
1002	B	50	30	74		

calculate total and avrg

DELETE

Friday, May 3, 2024 6:30 PM

DELETE:

- **It is used to delete the records.**

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

Syntax:

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

Deleting single record:

Delete an emp record whose empno is 7788:

```
DELETE FROM emp
WHERE empno=7788;

COMMIT;
```

Deleting specific group of records:

Delete all managers records:

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

Deleting all records:

Delete all emp records:

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

Examples on DELETE:

delete the emp records whose annual salary is more than 40000:

```
DELETE FROM emp
WHERE sal*12>40000;
```

delete all deptno 10 and 30 emps:

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

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

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

TCL

Friday, May 3, 2024 6:49 PM

TCL:

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

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

  - **A transaction must be successfully finished or cancelled.**
  - **Every transaction ends with either COMMIT or ROLLBACK.**

- **If transaction is successful, to save it use COMMIT.**
- **If transaction is unsuccessful, to cancel it use ROLLBACK.**

Example:

ACCOUNTS

ACNO	NAME	BALANCE
1001	A	80000-10000 = 70000
1002	B	40000+10000 = 50000

Transaction: Fund transfer
transfer 10000 amount from 1001 accout to 1002

sufficient funds available or not => SELECT
if available,
UPDATE from account balance => UPDATE
UPDATE to account balance => UPDATE

ORACLE SQL provides 3 TCL commands. They are:

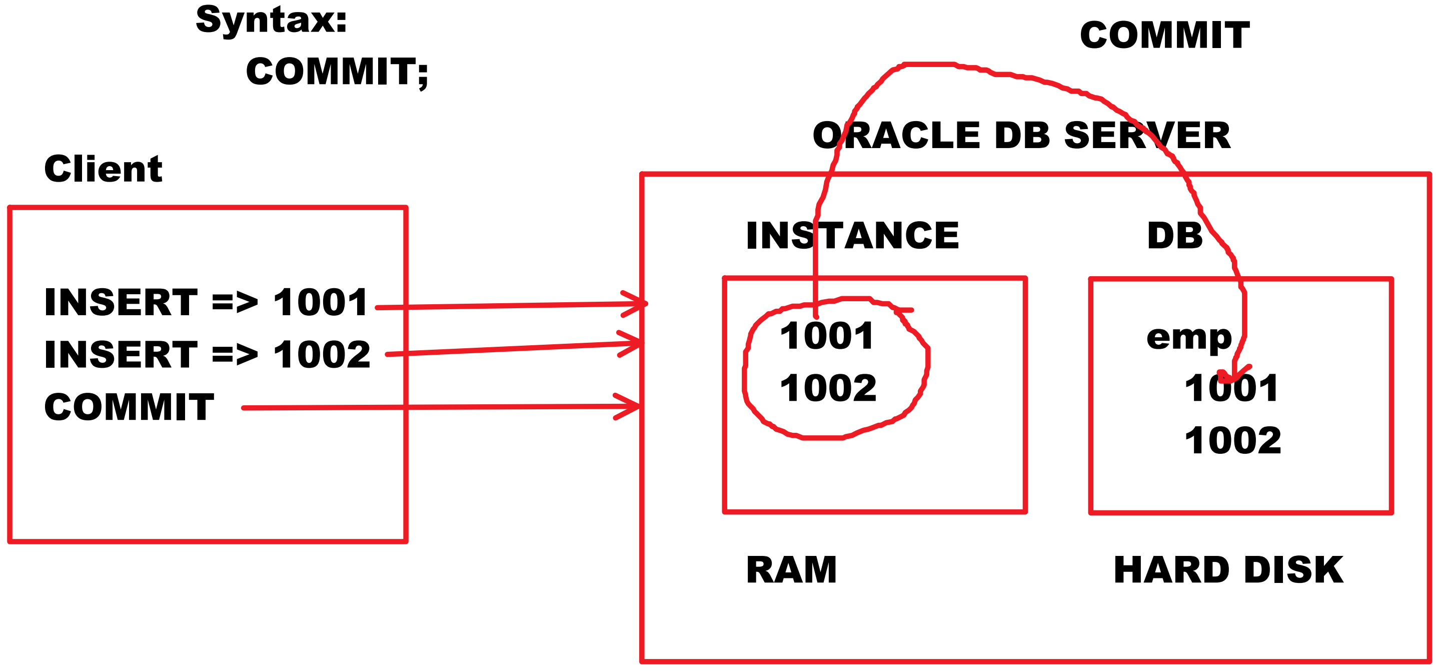
- COMMIT
- ROLLBACK
- SAVEPOINT

COMMIT [save]:

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

Syntax:

```
COMMIT;
```



ROLLBACK :

- ROLLBACK is used to cancel the transaction.

- **It cancels all uncommitted actions.**

Syntax:

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

Example on COMMIT and ROLLBACK:

STUDENT5

SID	SNAME

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

INSERT INTO student5 VALUES(1001,'A);
INSERT INTO student5 VALUES(1002,'B');
INSERT INTO student5 VALUES(1003,'C');
COMMIT;

SELECT * FROM student5;
Output:
1001   ..
1002   ..
1003   ..


INSERT INTO student5 VALUES(1004,'D');
INSERT INTO student5 VALUES(1005,'E');

SELECT * FROM student5;
Output:
1001
```

```
1002
1003
1004
1005

ROLLBACK;

SELECT * FROM student5;
Output:
1001
1002
1003
```

SAVEPOINT:

- **It is used to set margin [specific point] for rollback.**

Syntax:

```
SAVEPOINT <savepoint_name>;
```

Example:

```
BEGIN TRANSACTION t1      => 7.00 PM
   INSERT
   INSERT
   SAVEPOINT p1           => 7.10 PM
   INSERT
   INSERT
   SAVEPOINT p2           => 7.20 PM
   INSERT
   INSERT
   ROLLBACK TO p2;
```

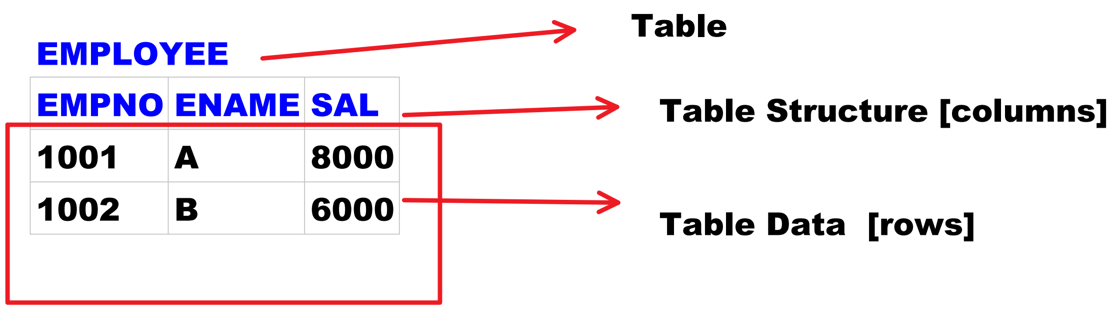
Note:
All DDL commands are auto committed.
All DML commands are not auto committed.

Example:

```
CREATE TABLE t1(f1 INT);      --CREATE+COMMIT => committed

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;    --2 actions cancelled
```

ALTER

Saturday, May 4, 2024 6:12 PM



EMPNO	ENAME	SAL
1001	A	8000
1002	B	6000

Table = structure [columns] + data [rows]

Note:

- **DDL commands deal with metadata**
- **DML commands deal with data**

ALTER:

- **ALTER => Change**

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

- **Using ALTER command we can:**

○ add the columns	=> ADD
○ rename the columns	=> RENAME COLUMN
○ drop the columns	=> DROP
○ modify the field sizes	=> MODIFY
○ modify the data types	=> MODIFY

Syntax:

```
ALTER TABLE <table_name> [ADD(<field_definitions>)]
```

Syntax:

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

Example on ALTER:

STUDENT

SID	SNAME

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

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

Adding a column [m1]:

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

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

Adding multiple columns [m2, m3]:

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

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

Renaming Column [m3 TO maths]:

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

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

Note:
using RENAME COLUMN, we can rename one column only

Dropping column [maths]:

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

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

Dropping multiple columns [m1, m2]:

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

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

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

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

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

can we decrease field size?
yes. we can decrease up to max string length in column

```
SNAME  VARCHAR2(20)
---------
```

SAI
NARESH   => max string length 6. we can decrease up to 6
KIRAN
RAJU

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

```
ALTER TABLE student MODIFY sid CHAR(8);

DESC student
Output:
NAME                TYPE
-----------------------------------------
SID                 CHAR(8)
SNAME               VARCHAR2(20)
```

Note:
To modify data type column must be empty

DROP, FLASHBACK, PURGE

Saturday, May 4, 2024 7:04 PM

DROP:

- **DROP command is used to drop [delete] entire table.**

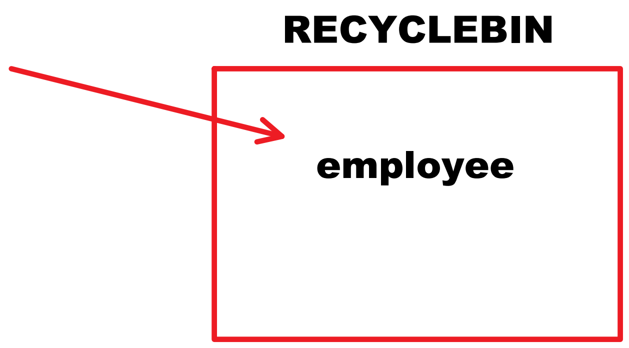
- **When we drop the table, it goes to RECYCLEBIN.**

Syntax:

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

Example:

```
DROP TABLE employee;
```



Note:

- **RECYCLEBIN feature added in ORACLE 10g version**

FLASHBACK:

- **introduced in ORACLE 10g version.**
- **it is used to restore the dropped table.**

Syntax:

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

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

PURGE:

- introduced in ORACLE 10g version.
- It is used to delete the table from RECYCLEBIN.

Syntax:
```
PURGE TABLE <table_name>;
```

Example:
```
PURGE TABLE employee;
--employee table will be deleted from recyclebin
--employee table deleted permanently
```

to see recyclebin:

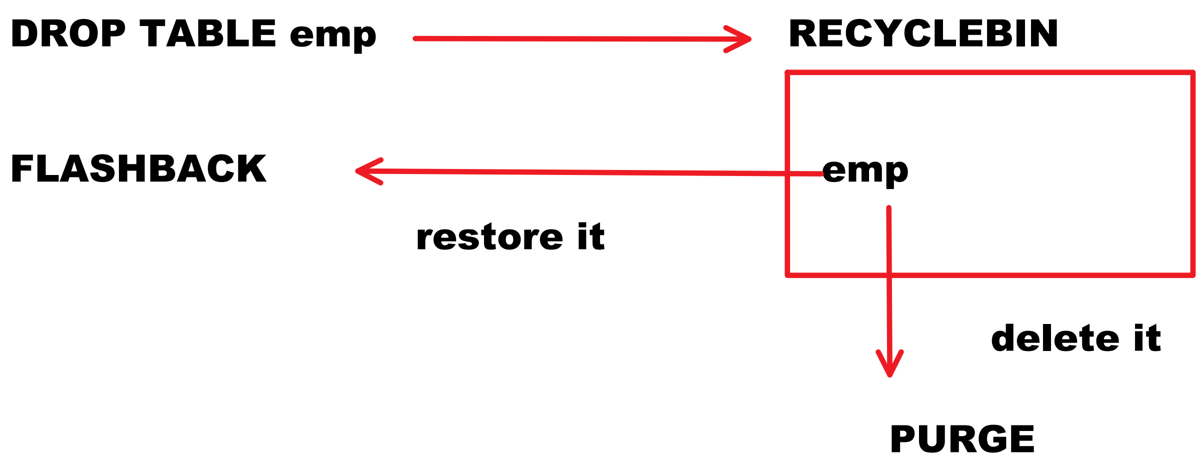
```
SHOW RECYCLEBIN
--displays dropped tables
```

to empty rcyclebin:

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

Note:
login as USER and practice DROP, FLASHBACK and PURGE

RECYCLEBIN will not work for SYSTEM user



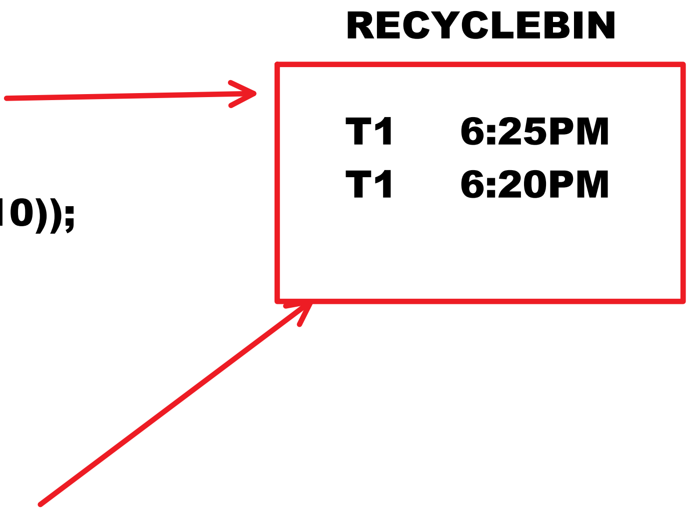
deleting a table permanent:

```
DROP TABLE t1;          --t1 will be moved to recyclebin
PURGE TABLE t1;

(or)

DROP TABLE t1 PURGE;    --t1 will be deleted permanently
```

CASE-1:

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

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

DROP TABLE t1;   --6:20 PM

CREATE TABLE t1(f1 VARCHAR2(10));

INSERt INTO t1 VALUES('A');
INSERt INTO t1 VALUES('B');
COMMIT;

DROP TABLE t1;    --6:25 PM
```

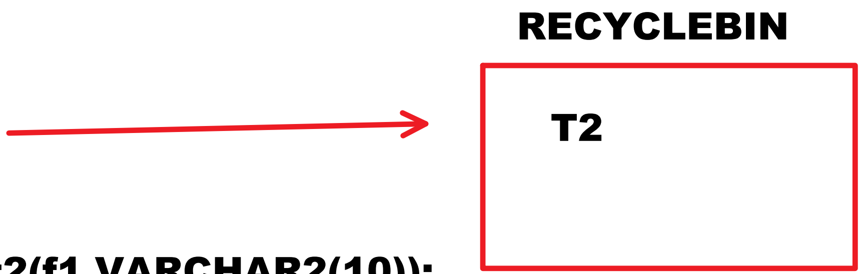


to restore older t1:

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

CASE-2:

```
CREATE TABLE t2(f1 int);

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

DROP TABLE t2;

CREATE TABLE t2(f1 VARCHAR2(10));
```



```
INSERT INTO t2 VALUES('A');
INSERT INTO t2 VALUES('B');
COMMIT;
```

```
FLASHBACK TABLE t2
TO BEOFRE DROP;
```

Output:

ERROR: original name used by existing object

```
FLASHBACK TABLE t2
TO BEFORE DROP
RENAME TO t2_old;
```

```
c##batch6pm
    T2
    T2_old
```

TRUNCATE, RENAME

Monday, May 6, 2024 6:47 PM

TRUNCATE:

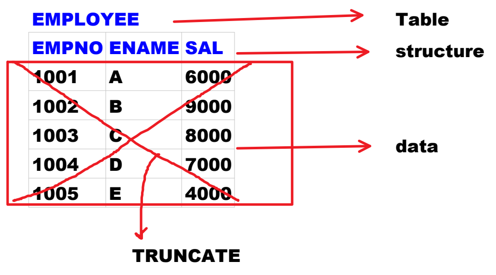
- **it is used to delete all rows with good performance.**

Syntax:

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

Example:

```
TRUNCATE TABLE employee;
```



EMPNO	ENAME	SAL
1001	A	6000
1002	B	9000
1003	C	8000
1004	D	7000
1005	E	4000

Differences b/w DROP and TRINCATE:

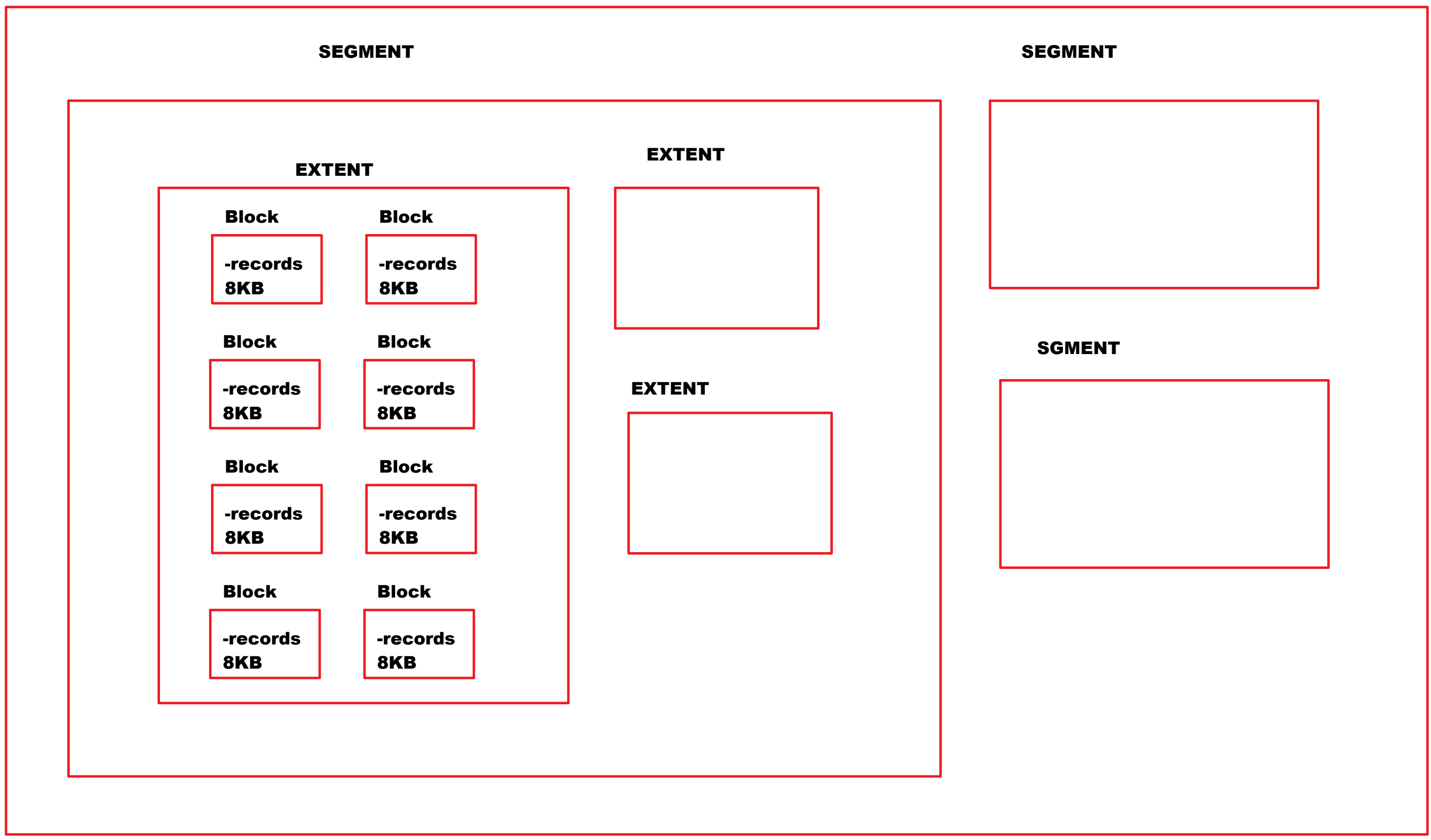
DROP	• **it is used to delete entire table** • **it deletes table structure also** • **it can be flashed back**
TRUNCATE	• **it is used to delete all rows** • **does not delete table structure** • **it cannot be flashed back**

```
DELETE FROM employee;          --deletes all records
TRUNCATE TABLE employee;       --deletes all records
```

Differences b/w DELETE and TRUNCATE:

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

TABLESPACE



TABLESPACE
- **SEGMENTS**
  - **EXTENTS**
    - **BLOCKS**
      - **RECORDS**

RENAME:
- **it is used to rename the table**

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

Example:
```
RENAME emp TO e;
```

Note:

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

DDL command = DDL command + COMMIT

```
CREATE = CREATE + COMMIT
ALTER   = ALTER + COMMIT
TRUNCATE = TRUNCATE + COMMIT
```

```
CREATE TABLE t3(f1 INT);            -- committed
INSERT INTO t3 VALUES(1);
INSERT INTO t3 VALUES(2);
CREATE TABLE t4(f1 VARCHAR2(10));    --committed
INSERT INTO t3 VALUES(3);
INSERT INTO t3 VALUES(4);
ROLLBACK;                  --cancels 2 actions
```

Monday, May 6, 2024 7:28 PM

DDL **metadata**	**DRL** **retrievals**	**DML** **data**	**TCL** **transactions**	**DCL** **accessibility**

CREATE
 	
ALTER
 	
DROP
FLASHBACK
PURGE
 	
TRUNCATE
 	
RENAME
	SELECT	**INSERT**
		UPDATE
		DELETE
		
		INSERT ALL
		MERGE
			COMMIT
			ROLLBACK
			SAVEPOINT
				GRANT
				REVOKE

DCL

Monday, May 6, 2024 7:34 PM

DCL / ACL:

- DCL => Data Control Language
- ACL => Accessing Control Language

- It deals with data accessibility.

- it is used to implement table level security.

- ORACLE SQL provides 2 DCL commands. They are:
  - GRANT
  - REVOKE

GRANT:

- it is used to give permission to other users
  on DB Objects like tables, views.

  Syntax:

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

Examples:

```
c##batch6pm   OWNER
   TABLE EMP
```

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

login as c##batch6pm:

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

login as c##userA:

```
SELECT * FROM c##batch6pm.emp;
```

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

login as c##batch6pm:

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

login as c##userA:

```
INSERT INTO c##batch6pm.emp(empno,ename,sal)
VALUES(1001,'A',6000);
Output:
1 row created.
```

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

login as c##batch6pm:

```
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 to all users [Making table as public]:

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

REVOKE:

- **It is used to cancel the permissions.**

Syntax:

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

Examples:

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

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

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

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

Example on GRANT and REVOKE:

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

Login as DBA:

```
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 windows side by side [press windows+right arrow]

c##userA [GRANTOR]

T1

F1	F2
1	A
2	B

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

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

c##userB [GRANTEE]

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

Output:
ERROR: Table does not exist

c##userA [GRANTOR]

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

c##userB [GRANTEE]

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

Output:

F1	F2
1	A
2	B

```
INSERT INTO c##userA.t1
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=1;
Output:
ERROR: insufficient privileges
```

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

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

```
select * from  t1;
Output:
```

F1	F2
1	A
2	B

```
COMMIT;   --after commit changes reflect to owner
```

```
select * from  t1;
Output:
```

F1	F2
1	A
2	B

2	B
3	C

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

COMMIT;

select * from t1;
Output:

F1	F2
1	sai
2	B
3	C

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

COMMIT;

select * from t1;
Output:

F1	F2
2	B
3	C

ALTER TABLE c#3userA.t1
ADD f3 DATE;
Output:
ERROR: insufficient privileges

GRANT all
ON t1
TO c##userB;

ALTER TABLE c#3userA.t1
ADD f3 DATE;
Output:
Table Altered

DESC t1
Output:
F1
F2
F3

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

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

SELECT *
FROM c#3userA.t1;
--displays data

REVOKE all
ON t1
FROM c##userb;

user_tables	it is a system table / readymade table it maintains all tables information

user_tab_privs_made	it is a system table / readymade table it maintains all permissions made by GRANTOR

user_tab_privs_recd	**it is a system table / readymade table** **it maintains all permissions recieved by GRANTEE**

to check permission received by GRANTEE:

c##userB:

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

to check permissions made by GRANTOR:

c##userA:

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

All = 12 permissions [by default]

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

Copying Tables and Copying Records

Wednesday, May 8, 2024 6:14 PM

Copying Tables and Copying Records

Copying Table:

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

Syntax:

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

Example-1:

Create exact copy of emp table with the name emp1:

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

Example-2:

Create a new table from existing table emp
with the name emp2
with 4 columns empno, ename, job, sal

with managers records

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

Copying table Structure:

Syntax:

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

false condition:
WHERE 1=2
WHERE 'A'='B'
WHERE 500=600

Example:

Create a new table emp3 with emp table structure without rows:

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

Copying records:

Syntax:

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

Example:

EMP
8 columns
13 rows

EMP4
8 columns
no rows

copy to emp4

Copy emp table all rows to emp4:

```
CREATE TABLE emp4
AS
SELECT * FROM emp
WHERE 1=2;
```

```
INSERT INTO emp4
SELECT * FROM emp;
```

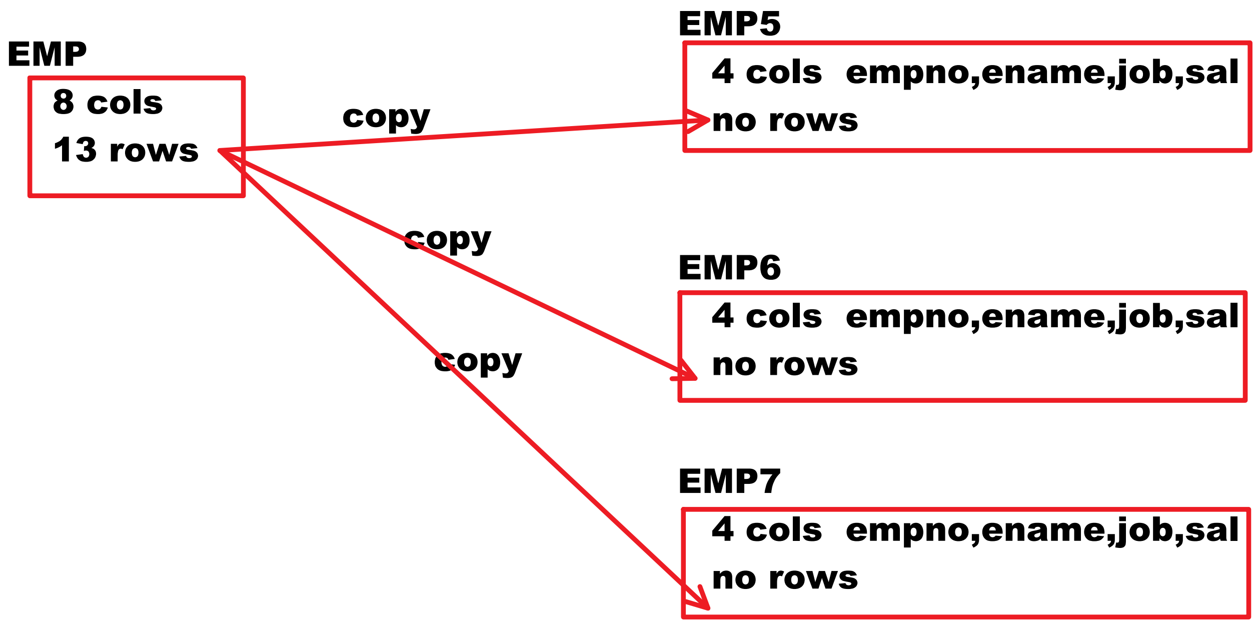
INSERT ALL

Wednesday, May 8, 2024 6:42 PM

INSERT ALL:

- **Introduced in ORACLE 9i version.**
- **It is used to copy one table records to multiple tables.**
- **it avoids of writing multiple INSERT commands.**
- **it can be used to perform ETL operations.**
  E => Extract T => Transfer L => Load

Example:


EMP
8 cols
13 rows
copy
copy
copy
EMP5
4 cols empno,ename,job,sal
no rows
EMP6
4 cols empno,ename,job,sal
no rows
EMP7
4 cols empno,ename,job,sal
no rows

INSERT ALL can be used in 2 ways. They are:

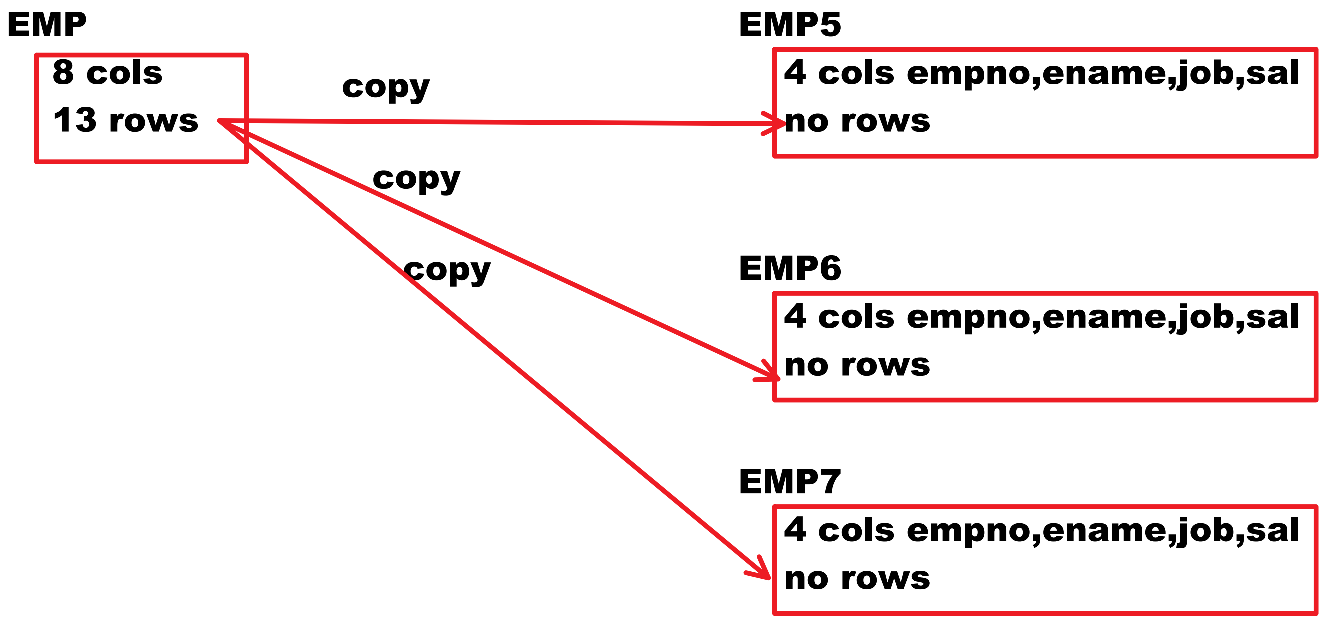
- **Unconditional INSERT ALL**
- **Conditional INSERT ALL**

Unconditional INSERT ALL:

Syntax:

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

Example on Unconditional INSERT ALL:


EMP
8 cols
13 rows
copy
copy
copy
EMP5
4 cols empno,ename,job,sal
no rows
EMP6
4 cols empno,ename,job,sal
no rows
EMP7
4 cols empno,ename,job,sal
no rows

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

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

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

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

Copy emp table all rows to emp5, emp6, emp7:

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

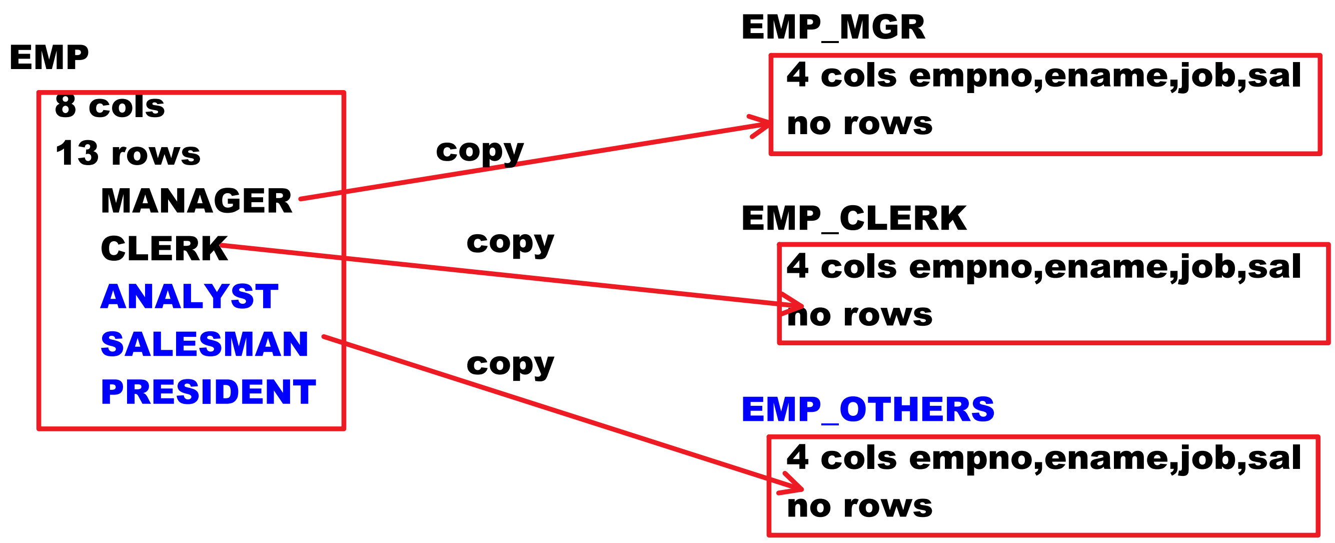
Conditional INSERT ALL:

Syntax:

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

<SELECT query>;

Example on conditional INSERT ALL:



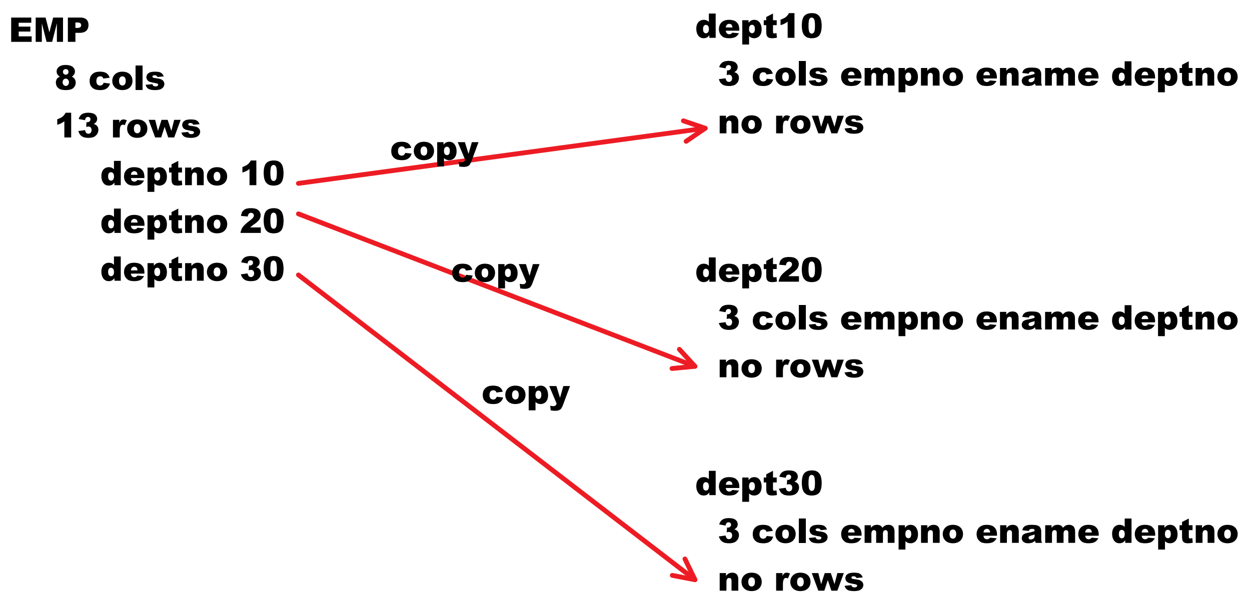
```
CREATE TABLE emp_mgr
AS
SELECT empno, ename, job, sal
FROM emp
WHERE 1=2;

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

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

copy all managers records to emp_mgr,

all clerks records to emp_clerk,
others copy to emp_others:

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

Assignment:



MERGE

Thursday, May 9, 2024 6:15 PM

MERGE:

Branch Office

CUSTOMER1

CID	CNAME	CCITY
1001	A AB	HYD BLR
1002	B	DLH
1003	C	MUM
1004	D	HYD
1005	E	VZG

Head Office

CUSTOMER2
[Replica => duplicate copy]

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

MERGE:

- **MERGE command introduced in ORACLE 9i version.**
- **MERGE command is used to apply one table changes to it's replica.**
- **MERGE = UPDATE + INSERT**
- **MERGE is a combination of UPDATE and INSERT.**
  - **if existing record, UPDATE it**
  - **if new record, INSERT it.**
- **It can be also called as "UPSERT" command.**
- **It avoids of writing a separate PL/SQL program.**

Syntax:

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

Example on MERGE:

s.cid = t.cid

CUSTOMER1 s

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

CUSTOMER2 t

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

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

1001	A	HYD
1002	B	DLH
1003	C	MUM

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

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

1004	D	HYD
1005	E	VZG

```
INSERT INTO customer1 VALUES(1004,'D','HYD');
INSERT INTO customer1 VALUES(1005,'E','VZG');
COMMIT;
```

1001	~~A~~ AB	~~HYD~~ BLR

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

COMMIT;
```

s.cid = t.cid

CUSTOMER1 s

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

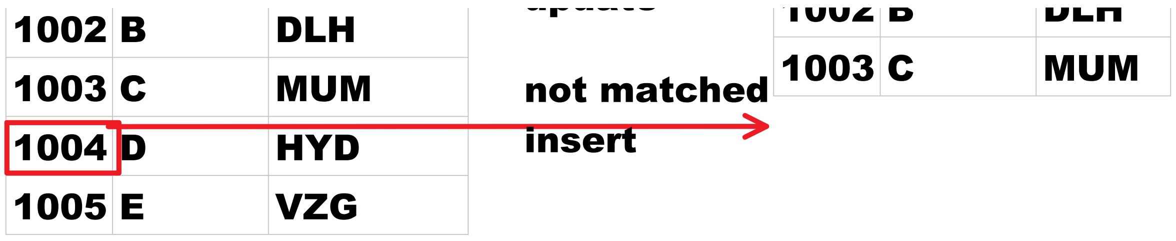
matched

update

not matched

CUSTOMER2 t => replica

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



1002	B	DLH
1003	C	MUM
1004	D	HYD
1005	E	VZG

1002	B	DLH
1003	C	MUM

Apply customer1 table changes to it's replica Customer2:

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

Output:

```
5 rows merged.
```

DUAL

Thursday, May 9, 2024 7:16 PM

DUAL:

- **DUAL is a readymade table**
- **It has 1 column, 1 row**

desc dual

select * from dual;

DUAL

DUMMY
X

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

Example:

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

dual table has 1 row. so we get one 300

```
SELECT 100+200 FROM emp;
Output:
300
300
300
.
.
```

emp table has 13 rows. so we get thirteen 300s
for every row 1 time 100+200 calculated

Note:
for windows os, latest version is ORACLE 21C

for LINUX os, latest version is ORACLE 23AI

In ORACLE 23AI, using FROM clause is optional.

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

Built-In Functions

Thursday, May 9, 2024 7:02 PM

Built-In Functions:

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

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

String Functions:

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

lower():

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

Syntax:

```
lower(<string>)
```

Examples:

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

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

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

Syntax:
upper(<string>)

Examples:

upper('raju')	RAJU

initcap() [initial capital]:
it is used to get every word's initial 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
ename=initcap(ename);
```

Display the emp record whose name is BLAKE:

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

```
ENAME           lower(ename)='blake'
--------------  ------------------------------------
SMITH           lower('SMITH')   smith = blake  F
ALLEN           lower('ALLEN')   allen  = blake  F
```

SMITH	lower('SMITH')	smith = blake	F
ALLEN	lower('ALLEN')	allen = blake	F
WARD	lower('WARD')	ward = blake	F
BLAKE	lower('BLAKE')	blake = blake	T

WHERE lower(ename)='blake';
(or)
WHERE upper(ename)='BLAKE';
(or)
WHERE initcap(ename) ='Blake';

length():

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

Syntax:

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

Examples:

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

Example:

Display the emp records whose names are having 4 chars:

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

(or)

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

Display the emp records whose names are having 14 chars:

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

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

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

Concat():

- **concatenate => combine**
- **it is used to combine 2 strings.**

Syntax:

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

Examples:

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

Example:

EMPLOYEE

EMPNO	FNAME	LNAME
1001	RAJ	KUMAR
1002	SAI	KRISHNA
1003	RAVI	TEJA

initcap(fname || ' ' || lname)

```
ENAME
-------------
Raj Kumar
Sai Krishna
Ravi Teja
```

```
CREATE TABLE employee
(
empno NUMBER(4),
fname VARCHAR2(10),
lname VARCHAR2(10)
);

INSERT INTO employee VALUES(1001,'RAJ','KUMAR');
INSERT INTO employee VALUES(1002,'SAI','KRISHNA');
INSERT INTO employee VALUES(1003,'RAVI','TEJA');
COMMIT;
```

Add NAME column:

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

concatenate first name and last name and store it in initcap case:

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

COMMIT;
```

Drop FNAME and LNAME columns:

```
ALTER TABLE employee DROP(fname, lname);
```

Substr():

- **it is used to get sub string from the string.**

Syntax:

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

Examples:

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

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

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

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

Substr('RAJ KUMAR',1,5)	RAJ K

2nd argument is position

position number can be -ve.

+ve	from left side
-ve	from right side

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

Examples:

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

EMPNO	ENAME	MAIL_ID
7369	SMITH	AMI369
7499	ALLEN	ALL499

Add mail_id column:

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

generate mail ids:

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

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

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

Display the emp records whose names are ended with S:

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

Display the emp records whose name's starting letter and ending letter are same:

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

Instr():

- **it is used to check whether sub string is existed in string or not.**

- **if sub string is existed, it returns position number.**
- **if sub string is not existed, it returns 0.**

Syntax:

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

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

Example:

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

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

1	2	3	4	5	6					11					16			
R	A	V	I		T	E	J	A		R	A	V	I		T	E	J	A

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

we can give search position [3rd arg] as -ve

+ve	from left side
-ve	from right side

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',-1,3)	6
Instr('THIS IS HIS WISH','IS',-4)	10

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

Examples on INSTR:

Display the emp records whose names are having AM letters:

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

(or)
```

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

Display the emp records whose names are having _ [underscore]:

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

(or)

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

(or)

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

Display the emp names whose names are having %:

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

(or)

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

Example:

EMPLOYEE

EMPNO	ENAME
1001	SAI KRISHNA
1002	RAVI TEJA
1003	KIRAN KUMAR

FNAME	LNAME
SAI	KRISHNA
RAVI	TEJA
KIRAN	KUMAR

```
CREATE TABLE employee
(
empno NUMBER(4),
ename VARCHAR2(20)
);

INSERT INTO employee VALUES(1001,'SAI KRISHNA');
INSERT INTO employee VALUES(1002,'RAVI TEJA');
INSERT INTO employee VALUES(1003,'KIRAN KUMAR');
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);

COMMIT;

ALTER TABLE employee DROP(ename);
```

Lpad() & Rpad():

- pad => fill
- L => Left
- R => Right

Lpad():

- it is used to fill specified char set at left side.

  Syntax:

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

3rd arg	default char	space

Rpad():

- **it is used to fill specified char set at right side.**

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

3rd arg	default char	space

Examples:

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

Lpad('A',6,'A')	AAAAAA
Lpad('X',8,'X')	XXXXXXXX

Display output as following:
Amount debited from acno XXXXXX7891
ACNO:1234567891

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

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

10-4 = 6

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

- **Trim => Remove**
- **These trim functions are used to remove unwanted characters.**

Ltrim():

- **it is used to remove unwanted characters from left side**

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

2nd arg	default char	space

Rtrim():

- **it is used to remove unwanted characters from right side**

Syntax:

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

2nd arg	default char	space

Trim():

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

Syntax:

Trim(LEADING/TRAILING/BOTH <char> FROM <string>)

Examples:

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

Trim(LEADING '*' FROM '***RAJU***')	RAJU***
Trim(TRAILING '*' FROM '***RAJU***')	***RAJU
Trim(BOTH '*' FROM '***RAJU***')	RAJU

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

Replace() and Translate():

Replace():

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

Syntax:

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

Examples:

Replace('SAI KRISHNA', 'SAI', 'RAMA')	**RAMA KRISHNA**

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

Translate():

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

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

Examples:

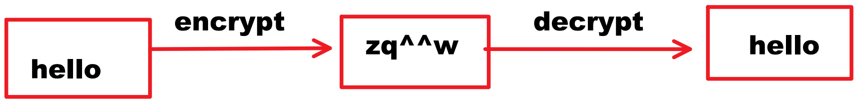
Replace('sai krishna', 'sai', 'XYZ')	**XYZ krishna**
Translate('sai krishna', 'sai', 'XYZ') **s => X** **a => Y** **i => Z**	**XYZ krZXhnY**

Replace('abcabcaabbccabc','abc','XYZ')	**XYZXYZaabbccXYZ**
Translate('abcabcaabbccabc','abc','XYZ')	**XYZXYZXXYYZZXYZ**

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

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

Note:
Translate() can be used to encrypt or decrypt the data.



normal text **cipher text** **normal text**

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

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

```
SELECT ename, translate(sal, '0123456789', 'z@#yqw*%b^') AS sal
FROM emp;
```

Reverse():
it is used to get reverse string

Syntax:
Reverse(<string>)

Example:

Reverse('ramu')	umar

ASCII():
it returns ASCII value of specified char

Examples:

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

Chr():
it returns char of specified ASCII value

Examples:

Chr(65)	A
Chr(97)	a

Soundex():
- **it is used to retrieve data based on sounds.**
- **when we don't know exact spelling then it is useful.**

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

Example:

Display blake record:

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

SELECT * FROM emp
WHERE Soundex(ename)=Soundex('BLEK');
```

Conversion Functions:

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".**

  Example:

  ```
  SELECT '10' + '20' FROM dual;
  ```

  $$\begin{tikzcd} \text{char} \arrow[d] & \text{char} \arrow[d] \\ \text{num} & \text{num} \end{tikzcd}$$

  Implicit conversion

  10 + 20

  ```
  Output:
  30
  ```

Note:

- **Don't depend on Implicit Conversion for 2 reasons:**
  - **Implicit conversion degrades the performance.**
  - **in further versions they may remove implicit conversion programs or they may modify implicit conversion programs.**

Explicit Conversion:

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

- **For explicit conversion we can use following conversion functions:**
  - **to_char()**
  - **to_date()**
  - **to_number()**

$$\begin{tikzcd}
\boxed{\text{DATE}} \arrow[r, bend left, "\text{to\_char()}"] & \boxed{\text{CHAR}} \arrow[l, bend left, "\text{to\_date()}"] \arrow[r, bend right, "\text{to\_number()}"'] & \boxed{\text{NUMBER}} \arrow[l, bend right, "\text{to\_char()}"']
\end{tikzcd}$$

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 covert date to char.**

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

Note:
sysdate function is used to get today's date

FORMAT	**PURPOSE**	**EXAMPLE sysdate: 16-MAY-24**	**OUTPUT**
YYYY	**year 4 digits**	**to_char(sysdate, 'YYYY')**	**2024**
YY	**year last 2 digits**	**to_char(sysdate, 'YY')**	**24**

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')**	**05**

MON	**short month name** **JAN, FEB**	**to_char(sysdate,'MON')**	**MAY**
MONTH	**full month name** **JANUARY**	**to_char(sysdate,'MONTH')**	**MAY**
D	**day num in week** **sun => 1** **mon => 2** **.** **.**	**to_char(sysdate,'D')**	**5**
DD	**day num in month**	**to_char(sysdate,'DD')**	**16**
DDD	**day num in year**	**to_char(sysdate,'DDD')**	**137**
DY	**short weekday name**	**to_char(sysdate,'DY')**	**THU**
DAY	**full weekday name**	**to_char(sysdate,'DAY')**	**THURSDAY**
Q	**quarter number** **jan to mar => 1** **apr to jun => 2** **jul to sep => 3** **oct to dec => 4**	**to_char(sysdate,'Q')**	**2**
CC	**century number**	**to_char(sysdate,'CC')**	**21**
HH / HH12	**hours part in 12hrs format**		
HH24	**hours part in 24 hrs format**		
MI	**minutes part**		
SS	**seconds part**		
FF	**fractional seconds**		
AM / PM	**AM or PM**		

Display current system date:

```
SELECT sysdate FROM dual;
```

Display current system time from sysdate:

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

Display current system date and time:

```
SELECT systimestamp FROM dual;
```

Display current system time from systimestamp:

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

Display all emp records along with hiredates.
Display hiredates in INDIA date format [DD/MM/YYYY]:

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

Display all emp records along with hiredates.
Display hiredates in US date format [MM/DD/YYYY]:

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

Display the emp records who joined in 1982:

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

(or)

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

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

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

Display the emp records who joined in december month:

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

(or)

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

Display the emp records who joined in january, may, december:

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

Display the emp records who joined in 4th quarter:

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

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

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

Display the emp records who joined in 1981 december month:

```
SELECT ename, hiredate
FROM emp
WHERE to_char(hiredate,'YYYY')=1981 AND to_char(hiredate,'MM')=12;
```

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

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

Display the emp records who joined on SUNDAY:

D	1
DY	SUN
DAY	SUNDAY

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

(or)

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

(or)

```
SELECT ename, hiredate
FROM emp
WHERE to_char(hiredate,'DAY')='SUNDAY';
Output:
no rows selected
```

```
to_char(hiredate,'DAY')='SUNDAY'
SUNDAY3spaces = SUNDAY  FALSE
```

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

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

(or)

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

WHERE to_char(hiredate,'DAY')='SUNDAY   ';

SUNDAY3spaces

To_Char() [number to char]:

- **To_Char() function can be used to convert number to char [string].**
- **To apply currency symbols, currency names, thousand separator ... 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'**

NLS	**National Language Support**

FORMAT	**PURPOSE**
L	**Currency Symbol** **default currency symbol is $**
C	**Currency name** **default currency name is USD**
9	**digit**
. / D	**decimal point**
, / G	**Thousand separator**

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

Display all emp names and salaries.

apply currency symbol $, thousand separator and 2 decimal places:

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

Note:

to see NLS PARAMETERS:

login as DBA:
  username: system
  password: nareshit

```
SQL> show parameters c  --displays all parameters

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

NLS_CURRENCY	$
NLS_ISO_CURRENCY	AMERICA

Display all emp names and salaries.
Apply japanese currency ¥:

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

Display all emp names and salaries.
Apply UK currency £:

```
SELECT ename, to_char(sal,'L99999.99','NLS_CURRENCY=£')  AS sal
FROM emp;
```

Display all emp names and salaries.
Apply INDIA currency RS:

```
SELECT ename, to_char(sal,'L99999.99','NLS_CURRENCY=RS')  AS sal
```

```
FROM emp;
```

Display all emp names and salaries.
Apply INDIA currency name:

```
SELECT ename,
to_char(sal,'C99999.99','NLS_ISO_CURRENCY=INDIA') AS sal
FROM emp;
```

to_date():

- **it is used to convert string to date.**
- **it is used to insert date values.**
- **extract part of specific date we can use it.**

Syntax:
```
to_date(<string>, <format>)
```

Example:

EMP1

EMPNO	ENAME	HIREDATE

```
CREATE TABLE emp1
(
empno NUMBER(4),
ename VARCHAR2(10),
hiredate DATE
);

INSERT INTO emp1 VALUES(1001, 'A', '27-DEC-2023');
```

String

Implicit Conversion

```
HIREDATE
-------------
27-DEC-23
```

DATE

```
INSERT INTO emp1 VALUES(1002, 'B', to_date('17-AUG-2020'));
                                            String
```

Explicit Conversion

```
HIREDATE
-----------------
17-AUG-20   DATE
```

```
INSERT INTO emp1
VALUES(1003,'C', to_date('17/10/2022','DD/MM/YYYY'));
```

```
INSERT INTO emp1
VALUES(1003,'C', to_date('&d/&m/&y','DD/MM/YYYY'));
Output:
enter  .. d: 20
enter .. m: 11
enter .. y: 2019

/
Output:
enter  .. d: 15
enter .. m: 8
enter .. y: 2018
```

to_date('25-dec-2023')	25-DEC-23
to_date('25 DECEMBER 2023')	25-DEC-23
to_date('DECEMBER 25 2023')	ERROR
to_date('DECEMBER 25 2023','MONTH DD YYYY')	25-DEC-23
to_date('25/12/2023')	ERROR
to_date('25/12/2023','DD/MM/YYYY')	25-DEC-23

Extract year part from the date '25-DEC-2023':

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

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

Find the weekday on which INDIA got independence:

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

SELECT to_char(to_date('15-AUG-1947'),'DAY')
FROM dual; date

On which weekday SACHIN born if SACHIN DOB is:
24-APR-1973:

```
SELECT to_char(to_Date('24-APR-1973'),'DAY')
FROM dual;
```

To_Number():

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

Syntax:

```
to_number(<string> [, <format>])
```

Examples:

to_number('123')	123
to_number('123.67')	123.67
to_number('$5000.00')	ERROR
to_number('$5000.00','L9999.99')	5000
to_number('USD5,000.00')	ERROR
to_number('USD5,000.00','C9,999.99')	5000
to_number('ABC')	ERROR

Aggregate Functions / Group Functions / Multi Row Functions:

ORACLE SQL provides following aggregate functions:

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

sum():
is used to find sum of a set of values

Syntax:

```
sum(<column>)
```

Examples:

Find sum of salaries of all emps:

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

Find sum of salaries of all managers:

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

Find sum of salaries of deptno 10:

```
SELECT sum(Sal) FROM emp
WHERE deptno=10;
```

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

Syntax:

```
Avg(<column>)
```

Examples:

Find average salary of all emps:

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

Find avg sal of all managers:

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

Min():
It is used to find minimum value in a set of values

Syntax:

Min(<column>)

Examples:

Find min sal in all emps:

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

Find min sal in deptno 30:

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

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

Syntax:
max(<column>)

Examples:

Find max salary in all emps:

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

Find max salary in all clerks:

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

count():
it is used to find number of records or number of column values.

Syntax:
count(* / <column>)

Examples:

Find no of records in emp table:

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

Find how many emps are getting commission:

SELECT count(comm) FROM emp;

difference b/w count(*) and count(<column>):

count(*)	**finds number of records**
count(<column>)	**finds number of column values**

difference between count(*) and count(<any number>):

```
SELECT count(*) FROM emp;          --displays no of records

SELECT count(25) FROM emp;        --displays no of records
SELECT count(30) FROM emp;        --displays no of records
SELECT count(8) FROM emp;        --displays no of records
```

count(*)	**it counts number of records slower**
count(8)	**it counts number of 8s faster**

Number Functions:

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

sqrt():

- **it us sued to find square root value**

Syntax:
sqrt(<number>)

Examples:

sqrt(100)	**10**

sqrt(81)	**9**

power():

- **it is used to find power values**

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

Examples:

power(5,3)	**125**
power(2,4)	**16**

sign():

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

Syntax:
sign(<number>)

Examples:

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

abs():

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

Syntax:
abs(<number>)

Examples:

abs(25)	**25**
abs(-25)	**25**

Mod():
it is used to get remainder value

Syntax:

Mod(<number>, <divisor>)

Examples:

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

ceil():

- it is used to get round up value

Syntax:
ceil(<number>)

floor():

- it is used to get round down value

Syntax:
round(<number>)

Examples:

ceil(123.6789)	123 and 124 124
floor(123.6789)	123

Trunc():

- it is used to remove decimal places.

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

Examples:

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

2nd argument can be given as -ve

if 2nd argument is -ve, it does not give decimal places

if 2nd argument is:

-1	rounds in 10s	10,20,30,40,...................

-2	rounds in 100s	100, 200, 300, ...
-3	rounds in 1000s	1000, 2000, 3000, ...

Trunc(123.5678,-1)	120 and 130 120
Trunc(786.45678,-1)	780 and 790 780
TRUNC(567.89234,-2)	500 and 600 500
TRUNC(4567.89543,-3)	4000 and 5000 4000
TRUNC(4567.89543,-1)	4560 and 4570 4560
TRUNC(4567.89543,-2)	4500 and 4600 4500

Round():

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

Syntax:

Round(<number>, <no_of_decimal_places>)

Examples:

TRUNC(123.678)	123
ROUND(123.678)	123 and 124 avrg: 123.5 124
TRUNC(123.478)	123
ROUND(123.478)	123 and 124 avrg: 123.5 123
Round(123.5)	123 and 124 avrg: 123.5 124
Trunc(123.5)	123

123+124 = 247
247/2 = 123.5

Trunc(123.6789,2)	123.67

Round(123.6789,2)	**123.68**
Round(123.6739,2)	**123.67**

Trunc(567.45678,3)	**567.456**
Round(567.45678,3)	**567.457**
Round(567.45648,3)	**567.456**

2nd argument can be given as -ve

if 2nd argument is -ve, it does not give decimal places

if 2nd argument is:

-1	**rounds in 10s**	**10,20,30,40,..................**
-2	**rounds in 100s**	**100, 200, 300, ...**
-3	**rounds in 1000s**	**1000, 2000, 3000, ...**

120+130 = 250
250/2 = 125

ROUND(127.5678,-1)	**120 and 130** **avrg: 125** **130**
TRUNC(127.5678,-1)	**120**

Round(567.8923,-2)	**500 and 600** **avrg: 550** **600**
Round(537.8923,-2)	**500 and 600** **avrg: 550** **500**

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

trunc() does not consider avrg. always gives lower value

round() considers avrg. if value is avrg or above avrg gives upper value. otherwise gives lower value

Date Functions:

sysdate
systimestamp

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

sysdate:
it returns current system date

systimestamp:
it returns current system date and time

Examples:

display current system date:

```
SELECT sysdate FROM dual;
```

display current system date and time:

```
SELECT systimestamp FROM dual;
```

Add_Months():

- **it is used to add months to specific date.**
- **it can be also used to subtract the months from specific date.**

Syntax:

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

Examples:

Add 2 days to today's date:

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

Add 2 months to today's date:

```
SELECT Add_Months(sysdate,2) FROM dual;
```

Add 2 years to today's date:

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

Subtract 2 days from today's date:

```
SELECT sysdate-2 FROM dual;
```

Subtract 2 months from today's date:

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

Subtract 2 years from today's date:

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

Examples:

ORDERS

ORDER_ID	CID	PID	QTY	PRICE	Ordered_Date	Delivery_Date
123456	..	..	..	..	sysdate	sysdate+5

PRODUCTS

PID	PNAME	MANUFACTURED_DATE	EXPIRY_DATE
1234	XYZ	sysdate	Add_Months(sysdate,3)

EMPLOYEE

EMPID	ENAME	DOBirth	DORetirement
1234	ABC	25-DEC-2000	Add_Months(DOBirth,60*12)

CMS_LIST

STATE_CODE	CM_NAME	START_DATE	END_DATE
TS	RR	9-DEC-2023	Add_Months(start_date,5*12)

```
INSERT INTO emp(empno,ename,hiredate)
VALUES(1001,'A',sysdate);

INSERT INTO emp(empno,ename,hiredate)
VALUES(1002,'B',sysdate-1);

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

```
VALUES(1003,'C',Add_months(sysdate,-1));

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

Display the emp records who joined today:

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

Output:
no rows selected.


hiredate                   =  sysdate
20-MAY-24 6:18 PM          =  20-MAY-24 6:26 PM    FALSE


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

TRUNC(hiredate)                 =  TRUNC(sysdate)
TRUNC(20-MAY-24 6:18 PM)        =  TRUNC(20-MAY-24 6:26 PM)
20-MAY-24                       =  20-MAY-24   TRUE

(or)

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

Display the emp records who joined yesterday:

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

Display the emp records who joined 1 month ago:

```
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:

SALES

DATEID	AMOUNT
1-JAN-23	50000
2-JAN-23	75000
..	
..	
20-MAY-24	60000

find today's sales:
where trunc(dateid) = trunc(sysdate)

find yesterday's sales:

find 1 month ago sales:

find 1 year ago sales:

Assignment:

GOLD_RATE

DATE_ID	PRICE
1-JAN-22	50000
2-JAN-22	53000
..	
20-May-24	70000

find today's gold rate

find yesterday's gold rate

find 1 month ago gold rate

find 1 year ago gold rate

Months_Between():

- **it is used to find difference between 2 date values.**
- **it returns number of months.**

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

Example:

Months_Between('20-MAY-2024','20-MAY-2023')	12
Months_Between('20-MAY-2024','20-MAY-2023')/12	1

Display all emp reocrds along with experience:

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

(or)

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

Display all emp records along wqith experience.
display experience in the form of years and months:

		year		months
15 months	trunc(15/12)	1	Mod(15,12)	3
30 months	trunc(30/12)	2	Mod(30,12)	6

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

Last_day():
it is used to get last date in the month.

Syntax:
Last_day(<date>)

Examples:

Last_day(sysdate)	31-MAY-24
Last_day('17-FEB-2024')	29-FEB-24

Last_day('17-FEB-2023')	**28-FEB-23**

Find next month first date:

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

Find current month 1st date:

```
SELECT
Last_day(Add_Months(sysdate,-1))+1 FROM dual;
              Last_day(20-APR-24)
                30-APR-24+1
```

Next_day():
it is used to get next date based on the weekday.
example: find next Friday date, find next sunday date

Syntax:

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

Examples:

find next Friday date:

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

find next Sunday date:

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

find next month first Sunday date:

```
SELECT next_day(last_day(sysdate),'sun') from dual;
```

find current month last Sunday date:

```
SELECT next_day(last_day(sysdate)-7,'sun') from dual;
```

Analytic Functions / Window Functions :

Rank()
Dense_Rank()
Row_Number()

dense = no gaps

MARKS
678
950
740
500
950
800
740
800
470
800

ORDER BY marks DESC

	RANK	DENSE_RANK
950	1	1
950	1	1
800	3	2
800	3	2
800	3	2
740	6	3
740	6	3
678	8	4
500	9	5
470	10	6

Rank():

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

Syntax:

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

Dense_Rank():

- **it is used to apply ranks to records according to specific column order.**
- **it follows sequence in ranking even if multiple values are same. it means, no gaps will be there in ranking.**

Syntax:

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

Examples:

Display all emp records along with salaries.
give top rank to highest salary:

```
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 emp names and hiredates.
apply ranks to records according to seniority.
give top rank to most senior:

```
SELECT ename, hiredate,
dense_rank() over(order by hiredate asc) AS rank
FROM emp;
```

Apply ranks to emp records according to salary is descending order.

```
SELECT ename, sal, hiredate,
dense_rank() over(order by sal desc, hiredate asc) as rank
FROM emp;
```

ORDER BY clause:

- **it is used to arrange the records in ascending or descending order.**

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

PARTITION BY clause:

- **it us used to group the records according to particular column**

Syntax:

PARTITION BY deptno

```
deptno
---------
10              10
10
10

20
20              20
20

30
30              30
30
```

Examples on PARTITION BY:

apply ranks to the records.
within dept apply ranks according to salary descending order.

```
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
```

Apply ranks to records.
with in job, according to salary descending order apply the ranks:

```
break on job skip 1 duplicates

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

clear breaks
```

Row_Number():

- **It is used to apply row numbers to records.**
- **On result of SELECT QUERY row numbers will be applied.**

Syntax:

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

Examples:

Apply row numbers to all emp records:

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

Apply row numbers to all emp records whose salaries are more than 2500:

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

Apply row numbers with in dept according to empno ascending order:

```
break on deptno skip 1 duplicates

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

Other Functions:

NVL()
NVL2()

Greatest()
Least()

User
UID

DECODE()

NVL() and NVL2():

NVL():

- **It is used to replace the nulls.**

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

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

Examples:

NVL(10,20)	10
NVL(null,20)	20

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

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

Display all emp records along with comm values.
If comm is null replace it with N/A [Not Applicable]:

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

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

Example:

STUDENT

SID	SNAME	M1 [number]
1234	A	78
1235	B	
1236	C	0
1237	D	66
1238	E	
1239	F	

replace nulls with ABSENT

nvl(to_char(m1),'ABSENT')

1240	G	44

Note:
NVL() function can replace nulls only

NVL2():

- **NVL2() function can 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

Example:

set comm as 700 to the emps who are not getting commission.
increase 1000 rupees comm to the emps who are getting comm:

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

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

NVL()	• it replaces nulls only • it takes 2 arguments
NVL2()	• it replaces nulls and not nulls • it takes 3 arguments

Max():
it is used to find max value in vertical values [column]

Syntax:
```
Max(<column>)
```

Greatest():

it is used to find max value in horizontal values [row]

Syntax:

greatest(<value1>, <v2>, ...)

Examples:

T1

```
F1
-------
10
90
20
```

SELECT max(f1) FROM t1;

Output:

90

SELECT greatest(f1,f2,f3) FROM t1;

T1

F1	F2	F3
10	90	40
45	30	60
80	55	77

greatest(f1,f2,f3)

greatest(10,90,40) => 90

greatest(45,30,60) => 60

greatest(80,55,77) => 80

Differences b/w max() and greatest()

max()	• can take 1 argument • multi row function • is used to find max value in vertical values
greatest()	• can take variable length arguments • single row function • is used to find max value in horizontal values

single row function:

1 function call applied on 1 row

multi row function:

1 function call applied on multiple rows

Min():

it is used to find min value in vertical values

Syntax:

Min(<column>)

Least():
it is used to find min value in horizontal values

Syntax:
Least(<value1>, <v2>, ...)

Examples:

```
T1
   F1
   -------
   10
   90
   20
```

SELECT min(f1) FROM t1;
Output:
10

SELECT least(f1,f2,f3) FROM t1;

T1

F1	F2	F3
10	90	40
45	30	60
80	55	77

least(f1,f2,f3)
least(10,90,40) => 10
least(45,30,60) => 30
least(80,55,77) => 55

Differences b/w min() and least():

min()	• can take 1 argument • multi row function • is used to find min value in vertical values
least()	• can take variable length arguments • single row function • is used to find min value in horizontal values

User:
it returns current user name

UID:
it returns current user id

Example:

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

Decode():

- **It is used to implement "IF .. THEN .. ELSIF" in SQL.**
- **it can check equality condition only.**

Syntax:

```
Decode(<column>,
          <value1>,<return_expression1>,
          <value2>,<return_expression2>,
          .
          .
          <else_return_expression>)
```

Example:

Display all emp details with job titles as following:

```
PRESIDENT => BIG BOSS
MANAGER   => BOSS
Others    => EMPLOYEE
```

```
SELECT ename,
Decode(Job,
         'PRESIDENT', 'BIG BOSS',
         'MANAGER','BOSS',
         'EMPLOYEE') AS job,
sal FROM emp;
```

Example:

Increase salary of all emps as following:

```
deptno 10   => increase 10% on sal
deptno 20   => increase 20% on sal
others      => increase 5% on sal
```

```
UPDATE emp
SET sal=
decode(deptno,
         10,sal+sal*0.1,
20,sal+sal*0.2,
sal+sal*0.05);
```

Built-In Functions:

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

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

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

Date	**add_months()**
	sysdate
	sytimestamp
	next_day()
	last_day()

Analytic	**rank()**
	dense_rank()
	row_number()

Other	**NVL() NVL2()**
	Decode()
	user
	uid
	greatest()
	least()

CLAUSES

Wednesday, May 22, 2024 6:49 PM

SQL
QUERIES
CLAUSES

ENGLISH
SENTENCES
WORDS

CLAUSES of SELECT command:

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

Syntax of SELECT command: [oracle 21c]

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

SELECT Command Clauses are:

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

Display emp names and salaries of the emps whose salasries are more than 3000:

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

SELECT clause	**It is used to specify columns list**
FROM clause	**It is used to specify tables list**
WHERE clause	**It is used to filter condition**

ORDER BY:

- **ORDER BY clause is used to arrange the records in ascending or descending order according to specific column(s).**
- **default order is: ASC**

Syntax:

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

NUMBER		**CHAR**	
ASC	**DESC**	**ASC**	**DESC**
1	**10**	**A**	**Z**
2	**9**	**B**	**Y**
3	**8**	**.**	**X**
.	**.**	**.**	**.**
.	**.**	**Z**	**.**
10	**1**		**A**

Note:
Calendar order is ASCENDING ORDER.

ASC **DESC**

ASC	DESC
1-Jan-23	31-DEC-23
2-JAN-23	30-DEC-23
3-JAN-23	.
.	.
.	1-JAN-23
31-DEC-23	

Examples on ORDER BY:

Display all emp records.
arrange them in descending order according to sal:

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

ename	1
sal	2

(or)

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

(or)

```
SELECT *
FROM emp
ORDER BY 6 DESC;
```

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

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

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

(or)

SELECT ename, sal
FROM emp
ORDER BY 1 ASC;

(or)

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

Display all emp records.
Arrange them according to seniority. Display senior record first:

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

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

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

Arranging records in ascending or descending order according to multiple columns:

Arrange deptnos in ascending order.
with in dept arrange salaries in descending order:

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

CASE-1: deptnos are different => checks only deptno.
salary will not be checked

```
20          10
10          20
```

CASE-2: deptnos are same => if deptnos same then only it
checks salary

```
10    6000          10  8000
10    8000          10  6000
```

Display all emp records.
Arrange them in ascending order according to deptno.
Within dept, arrange them in ascending order according
to hiredate:

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

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

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

Note:
In ASCENDING ORDER, nulls will be displayed at last.
In DESCENDING ORDER, nulls will be given first.

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

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

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

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

GROUP BY:

- **GROUP BY is used to group the records according to particular column(s).**
- **On these groups, we apply aggregate functions.**
- **It is mainly used for data analysis.**
- **It gives summarized data from detailed data**

Syntax:

```
GROUP BY <grouping_columns_list>
```

Example:

```
GROUP BY deptno
```

GROUP BY deptno

emp => detailed data $\longrightarrow$ **summarized data**

EMPNO	ENAME	SAL	DEPTNO
1001	A	6000	10
1002	B	5000	10
1003	C	9000	10
1004	D	8000	20
1005	E	12000	20
1006	F	10000	30
1007	G	11000	30

deptno	sum(sal)
10	20000
20	20000
30	21000

Examples on GROUP BY:

Find dept wise sum of salaries:

DEPTNO	SUM_OF_SAL
10	?
20	?
30	?

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

Find sum of salaries of deptno 20 and 10:

DEPTNo	SUM_OF_SAL
10	..
20	..

```
SELECT deptno, sum(Sal) AS sum_of_Sal
FROM emp
WHERE deptno IN(10,20)
GROUP BY deptno
```

Execution order:

FROM
WHERE
GROUP BY
HAVING

WHERE deptno IN(10,20)
GROUP BY deptno
ORDER BY deptno ASC;

WHERE
GROUP BY
HAVING
SELECT
DISTINCT
ORDER BY
OFFSET
FETCH

emp

EMPNO	ENAME	SAL	DEPTNO
1004	D	8000	20
1005	E	12000	20
1006	F	10000	30
1007	G	11000	30
1001	A	6000	10
1002	B	5000	10
1003	C	9000	10

FROM emp:
it selects entire table

emp

EMPNO	ENAME	SAL	DEPTNO
1004	D	8000	20
1005	E	12000	20
1006	F	10000	30
1007	G	11000	30
1001	A	6000	10
1002	B	5000	10
1003	C	9000	10

WHERE deptno IN(10,20):
- **it filters the records**
- **WHERE condition will be applied on every**

EMPNO	ENAME	SAL	DEPTNO
1004	D	8000	20
1005	E	12000	20

1001	A	6000	10
1002	B	5000	10
1003	C	9000	10

GROUP BY deptno:
it groups the records according to specified column

1004	D	8000	20
1005	E	12000	20

sum(sal) => 20000

1001	A	6000	10
1002	B	5000	10
1003	C	9000	10

sum(sal) => 20000

SELECT deptno, sum(Sal) AS sum_of_Sal:

it selects the data

DEPTNO	Sum_OF_SAL
20	20000
10	20000

ORDER BY deptno ASC:
it arranges result in the order

DEPTNO	Sum_OF_SAL
10	20000
20	20000

Find dept wise number of emps:

DEPTNO	NO_OF_EMPS
10	?
20	?
30	?

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

Find Dept wise max salary and min salary:

DEPTNO	MAX_SAL	MIN_SaL
10	?	?
20	?	?
30	?	?

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

Find year wise no of emps joined in organization:

YEAR	NO_OF_EMPS
1980	?
1981	?
1982	?
1983	?

```
SELECT to_char(hiredate,'YYYY') AS year,
count(*) AS no_of_emps
FROM emp
GROUP BY to_char(hiredate,'YYYY')
ORDER BY 1;
```

Find quarter wise no of emps joined in organization:

QUARTER	NO_OF_EMPS
1	?
2	?
3	?
4	?

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

Assignment:
Find month wise no of emps joined in organization
GROUP BY to_char(hiredate,'MM')   MM/MON/MONTH

Find weekday wise no of emps joined in organization
GROUP BY to_char(hiredate,'D')   D / DY / DAY

Find job wise sum of salaries:

JOB	SUM_OF_SAL
MANAGER	?
CLERK	?

ANALYST	**?**

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

Find job wise max sal and min sal:

```
SELECT job, max(Sal) AS max_Sal,
min(sal) AS min_sal
FROM emp
GROUP BY job;
```

Grouping records according to multiple columns:

Find dept wise, job wise no of emps:

DEPTNO	JOB	NO_OF_EMPS
10	CLERK	?
10	MANAGER	?
20	CLERK	?
20	MANAGER	?

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

Rollup() and Cube():

- **These functions are used to calculate sub totals and grand total.**
- **we call these functions from GROUP BY clause.**

We pass grouping columns list as arguments.

Syntax:
```
GROUP BY Rollup(grouping_columns_list)
```

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

Syntax:
```
GROUP BY Cube(grouping_columns_list)
```

Example:
```
GROUP BY Cube(deptno, job)
```

Find dept wise, job wise no of emps.
Calculate sub totals and grand total according to deptno.
[Rollup()]

DEPTNO	JOB	NO_OF_EMPS
10	CLERK	?
10	MANAGER	?
	10th dept sub total	?
20	CLERK	?
20	MANAGER	?
	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, job wise no of emps.
Calculate sub totals and grand total according to deptno and job.
[Cube()]

DEPTNO	JOB	NO_OF_EMPS
10	CLERK	?

10	MANAGER	?
	10th dept sub total	?
20	CLERK	?
20	MANAGER	?
	20th dept sub total	?
	CLERK sub total	?
	MANAGER sub total	?
	GRAND TOTAL	?

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

Find dept wise, job wise sum of salaries:

DEPTNO	JOB	SUM_OF_SAL
10	CLERK	?
10	MANAGER	?
20	CLERK	?
20	MANAGER	?

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

Find dept wise, job wise sum of salaries.
Calculate sub totals and grand total according to deptno.
[Rollup()]

DEPTNO	JOB	SUM_OF_SAL

10	CLERK	?
10	MANAGER	?
	10th dept sub total	?
20	CLERK	?
20	MANAGER	?
	20th dept sub total	?
	GRAND TOTAL	?

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

Find dept wise, job wise sum of salaries.
Calculate sub totals and grand total according to deptno and job.
[Cube()]

DEPTNO	JOB	SUM_OF_SAL
10	CLERK	?
10	MANAGER	?
	10th dept sub total	?
20	CLERK	?
20	MANAGER	?
	20th dept sub total	?
	CLERK sub total	?
	MANAGER sub total	?
	GRAND TOTAL	?

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

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

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

```
SELECT to_char(hiredate,'YYYY') AS year,
to_char(hiredate,'Q') AS quarter,
count(*) AS no_of_emps
FROM emp
GROUP BY to_char(hiredate,'YYYY'), to_char(hiredate,'Q')
ORDER BY 1;
```

Find year wise, quarter wise no of emps joined in organization. calculate sub totals and grand toatl according to year [Rollup()]:

YEAR	QUARTER	NO_OF_EMPS
1980	1	?
	2	?
	3	?
	4	?
	1980 sub total	?
1981	1	?
	2	?
	3	?
	4	?
	1981 sub total	?
	GRAND TOTAL	?

```
SELECT to_char(hiredate,'YYYY') AS year,
to_char(hiredate,'Q') AS quarter,
    count(*) AS no_of_emps
    FROM emp
    GROUP BY Rollup(to_char(hiredate,'YYYY'), to_char(hiredate,'Q'))
    ORDER BY 1;
```

Find year wise, quarter wise no of emps joined in organization. calculate sub totals and grand toatl according to year and quarter [Cube()]:

YEAR	QUARTER	NO_OF_EMPS
1980	1	?
	2	?
	3	?
	4	?
	1980 sub total	?
1981	1	?
	2	?
	3	?
	4	?
	1981 sub total	?
	1st 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

find year wise, quarter wise sales

SALES

DATEID	AMOUNT
1-JAN-2020	50000
2-JAN-2020	75000
..	
..	
..	
24-MAY-2024	60000

find year wise, quarter wise sales. calculate sub totals and grand total according to year and quarter [cube()]

2020	1	?
	2	?
	3	?
	4	?
	2020 sales	?
2021	1	?
	2	?
	3	?
	4	?
	2021 sales	?
	1st qrtr sales	?
	2nd	
	3rd	
	4th	
	GRAND TOTAL	

Note:
WHERE clause is used to specify condition on rows

HAVING:
- **It is used to specify conditions 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 10000 rupees amount on their emps:

```
SELECT deptno, sum(sal)
FROM emp
GROUP BY deptno
HAVING sum(sal)>10000
ORDER BY 1;
```

Display the depts which are having 5 or more emps:

```
SELECT deptno, count(*)
FROM emp
GROUP BY deptno
HAVING count(*)>=5
ORDER BY 1;
```

Differences b/w WHERE and HAVING:

WHERE	HAVING
• WHERE condition will be applied on every row.	• HAVING condition will be applied on every group.
• it filters the rows	• it filters the groups
• it can be used without GROUP BY	• it cannot be used without GROUP BY
• we cannot use aggregate function in WHERE clause	• we can use aggregate function in HAVING clause
• It gets executed before GROUP BY	• It gets executed after GROUP BY

Can we use column alias in GROUP BY? Why?
NO. Because, GROUP BY gets executed before SELECT. [ORACLE 21C]

Can we use column alias in ORDER BY? Why?
YES. ORDER BY gets executed after SELECT. [ORACLE 21C]

Execution Order:

FROM
WHERE
GROUP BY
HAVING
SELECT
DISTINCT
ORDER BY
OFFSET
FETCH

find year wise no of emps:

```
SELECT to_char(hiredate,'YYYY') AS year,
count(*) AS no_of_emps
FROM emp
GROUP BY year;
```

Output:

```
ERROR: YEAR invalid identifier
```

```
SELECT to_char(hiredate,'YYYY') AS year,
count(*) AS no_of_emps
FROM emp
GROUP BY to_char(hiredate,'YYYY')
ORDER BY year;
```

Note:
In ORACLE 23AI version,
we can use column alias in GROUP BY

```
SELECT to_char(hiredate,'YYYY') AS year,
count(*) AS no_of_emps
FROM emp
GROUP BY year;
```

In 21c
ERROR

```
SELECT to_char(hiredate,'YYYY') AS year,
count(*) AS no_of_emps
FROM emp
GROUP BY year;
```

In 23AI
displays year wise no of emps

Note:
Till ORACLE 21C, we cannot use column alias in GROUP BY, HAVING, WHERE.

In ORACLE 23AI, we can use column alias in GROUP BY and HAVING.
We cannot use column alias in WHERE.

DISTINCT:

- **It is used to eliminate the duplicate records.**

Syntax:

```
SELECT DISTINCT <columns_list>
```

Examples on DISTINCT:

Display the job titles offered by company:

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

JOB
-----------
CLERK
SALESMAN
SALESMAN
ANALYST
MANAGER
ANALYST
MANAGER
MANAGER
CLERK
CLERK
```

```
SELECT DISTINCT job FROM emp;

JOB
-----------
CLERK
SALESMAN
ANALYST
MANAGER
```

Display the deptnos which are having employees:

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

DEPTNO
-------------
20
30
30
10
20
10
20
```

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

DEPTNO
-------------
10
20
30
```

```
30
10
20
```

Display dept wise, job titles offered by company:

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

10 CLERK
10 MANAGER
10 RESIDENT

20  CLERK
20  CLERK
20  MANAGER
20  ANALYST
20  ANALYST

30  CLERK
30  MANAGER
30  SALESMAN
30  SALESMAN
30  SALESMAN
30  SALESMAN
```

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

10 CLERK
10 MANAGER
10 RESIDENT

20  CLERK
20  MANAGER
20  ANALYST

30  CLERK
30  MANAGER
30  SALESMAN
```

OFFSET:
- **introduced in ORACLE 12c**
- **it is used to specify number of rows to be skipped.**

  Syntax:
  ```
  OFFSER <number> ROW/ROWS
  ```

FETCH:
- **introduced in ORACLE 12c**
- **it is used to specify number of rows to be fetched [selected]**

Syntax:

```
FETCH FIRST/NEXT <number> ROW/ROWS ONLY
```

Examples:

Display all emp table rows except first 5 rows:

```
SELECT * FROM emp
OFFSET 5 ROWS;
```

Display first 5 rows only:

```
SELECT * FROM emp
FETCH FIRST 5 ROWS ONLY;
```

Display 6th row to 10th row from emp table:

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

Execution Order [ORACLE 21C]:

FROM	used to specify tables list	FROM emp
		FROM emp, dept

WHERE	used to specify filter condition it filters the rows this condition will be applied on every row	WHERE sal>3000
GROUP BY	used to group the records	GROUP BY deptno
HAVING	used to write condition on groups it filters the groups it will be applied on every group	HAVING sum(Sal)>10000
SELECT / SELECT ALL	used to specify columns list	SELECT ename, sal

DISTINCT	**used to eliminate the duplicates**	**DISTINCT job**

ORDER BY	**to arrange records in ASC or DESC**
	
	default order: ASC
		ORDER BY sal DESC
		
		ORDER BY ename ASC

OFFSET	**used to specify no of rows to be skipped**	**OFFSET 5 ROWS**
FETCH	**used to specify no of rows to be fetched**	**FETCH FIRST 5 ROWS ONLY**

JOINS

Saturday, May 25, 2024 6:55 PM

Goal:

JOINS concept is used to retrieve the data from multiple tables

COLLEGE Database

Course
Student
Marks
Fee
Staff

S.SID = M.SID

STUDENT S

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

MARKS M

SID	Maths	Phy	Che
1001	70	90	60
1002	55	67	39
1003	48	72	81

JOINS

SID	SNAME	Maths

STUDENT **MARKS**

JOINS:

- **JOIN => connect / combine**
- **JOIN is an operation.**
- **In Join Operation, one table record will be joined with another table record based on some condition. This condition is called "Join Condition".**
- **Based on Join condition Join operation will be performed.**
- **Join condition decides which record in one table should be joined with which record in another table.**
- **JOINS concept is used to retrieve the data from multiple tables**

Types of Joins:

- **Inner Join** **= matched records only**
  - **Equi Join**
  - **Non-Equi Join**
- **Outer Join** **= matched + unmatched records**
  - **Left Outer Join**
  - **Right Outer Join**
  - **Full Outer Join**
- **Self Join**
- **Cross Join**

Inner Join:

- **Inner Join can give matched records only**

  2 Types:
  - **Equi Join**
  - **Non-Equi Join**

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

  Example:
  WHERE S.SID = M.SID

Example on Equi Join:

STUDENT S

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

MARKS M

SID	Maths	Phy	Che
1001	70	90	60
1002	55	67	39
1003	48	72	81

```
CREATE TABLE student
(
sid NUMBER(4),
sname VARCHAR2(10),
scity CHAR(3)
);

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

CREATE TABLE marks
(
sid NUMBER(4),
maths NUMBER(3),
phy NUMBER(3),
che NUMBER(3)
);

INSERT INTO marks VALUES(1001,70,90,60);
INSERT INTO marks VALUES(1002,55,67,39);
INSERT INTO marks VALUES(1003,48,72,81);
COMMIT;
```

Display student details with maths subject marks:

SID	SNAME	MATHS

STUDENT s MARKS m

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

Above query degrades the performance.

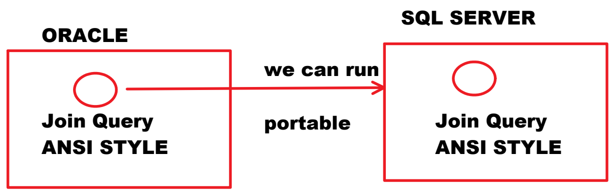
```
SELECT student.sid, student.sname, marks.maths
FROM student, marks
WHERE student.sid = marks.sid;
```

To make table name short use table alias.

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

Note:
For ORACLE 9i version, we can write Join Query in 2 styles. They are:

- **ORACLE STYLE / NATIVE STYLE**
- **ANSI STYLE   => Best way => portable**



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, we write Join Condition in WHERE clause.**
- **In ANSI STYLE, we write Join Condition in ON clause.**

Display student details along with maths subject marks:

SID	SNAME	MATHS

STUDENT s   MARKS m

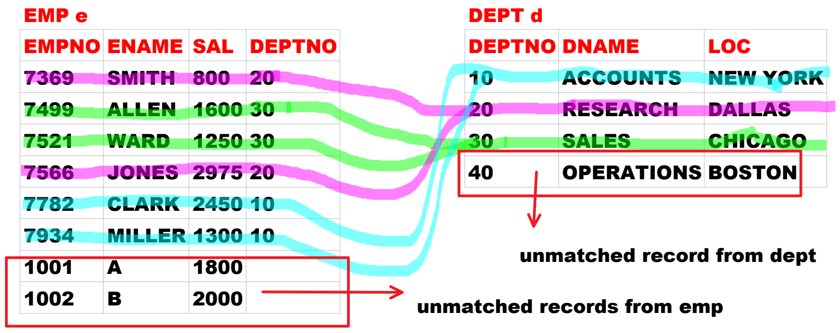
ORACLE STYLE:

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

ANSI STYLE:

```
SELECT s.sid, s.sname, m.maths
FROM student s INNER JOIN marks m
ON s.sid=m.sid;
```

Example: **e.deptno = d.deptno**



EMP e

EMPNO	ENAME	SAL	DEPTNO
7369	SMITH	800	20
7499	ALLEN	1600	30
7521	WARD	1250	30
7566	JONES	2975	20
7782	CLARK	2450	10
7934	MILLER	1300	10
1001	A	1800	
1002	B	2000	

DEPT d

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

unmatched record from dept

unmatched records from emp

Display emp details along with dept details as following:

ENAME	SAL	DNAME	LOC

EMP e DEPT d

ORACLE STYLE:

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

ANSI STYLE:

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

Display the emp details along with dept details.
Display the emps who are working in NEW YORK only:

ename	sal	dname	loc
			NEW YORK

to see execution plan:

```
SET AUTOTRACE ON EXPLAIN
```

ORACLE STYLE:

```
SEELCT e.ename, e.sal, d.dname, d.loc
FROM emp e, dept d
WHERE e.deptno=d.deptno AND d.loc='NEW YORK';
```

ANSI STYLE:

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

Note:
ON clause is used to specify Join Condition
WHERE clause is used to specify filter condition

e.deptno=d.deptno

EMP e

EMPNO	ENAME	SAL	DEPTNO
7369	SMITH	800	20
7499	ALLEN	1600	30
7521	WARD	1250	30
7566	JONES	2975	20
7782	CLARK	2450	10
7934	MILLER	1300	10
1001	A	1800	
1002	B	2000	

DEPT d

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

Note:
First filter condition will be executed.
Then join operation will be performed.

Display WARD record along with dept details as following:

ename	sal	dname	loc
WARD			

ORACLE STYLE:

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

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='WARD';
```

e.deptno = d.deptno

EMP e

EMPNO	ENAME	SAL	DEPTNO
7369	SMITH	800	20
7499	ALLEN	1600	30
7521	WARD	1250	30
7566	JONES	2975	20
7782	CLARK	2450	10
7934	MILLER	1300	10
1001	A	1800	
1002	B	2000	

DEPT d

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

Display emp records along with dept details.
Display the emps who are working in SALES dept:

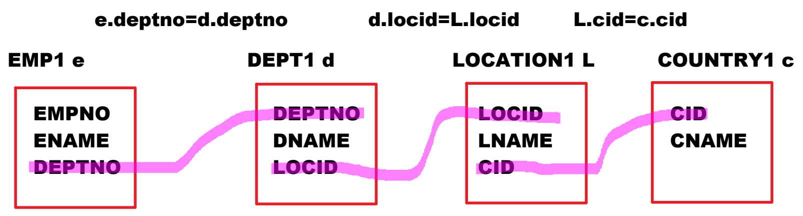
ORACLE STYLE:

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

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.dname='SALES';
```

Example:
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;
```

EMP1

EMPNO	ENAME	DEPTNO
1001	A	10

DEPT1

DEPTNO	DNAME	LOCID
10	SALES	100

LOCATION1

LOCID	LNAME	CID
100	HYD	123456

COUNTRY1

CID	CNAME
123456	INDIA

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

Examples:

```
WHERE s.sid = m.sid
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:

e.sal BETWEEN s.losal AND s.hisal

EMP e

EMPNO	ENAME	SAL
1001	A	1300
1002	B	7000
1003	C	6000
1004	D	1000
1005	E	2500

SALGRADE s

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

Display emp details along with salary grades:

ENAME	SAL	GRADE

emp e salgrade s

ORACLE STYLE:

```
SELECT e.ename, e.sal, s.grade
```

FROM emp e, salgrade s
WHERE e.sal BETWEEN s.losal AND s.hisal;

ANSI STYLE:

SELECT e.ename, e.sal, s.grade
FROM emp e INNER JOIN salgrade s
ON e.sal BETWEEN s.losal AND s.hisal;

Outer Join:

- **INNER JOIN can give matched records only.**
  To get unmatched records also we use OUTER JOIN.

- **INNER JOIN = matched records only**

- **OUTER JOIN = matched + unmatched records**

- **OUTER JOIN can give matched records and unmatched records.**

- **It has 3 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

  Example:
  WHERE e.deptno = d.deptno

emp e	**Left table**
dept d	**Right Table**

  WHERE d.deptno = e.deptno

dept d	**Left table**
emp e	**Right Table**

- **In ANSI STYLE,**
  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	**Right table**

Left Outer Join:

- **Left Outer Join = matched + unmatched from left table**

- **Left Outer join can give matched records and unmatched records from left table.**

- In ORACLE STYLE, write outer join operator (+) at right side.

- In ANSI STYLE, we use the keyword: LEFT [OUTER] JOIN

Example on Left outer join:

Display emp details along with dept details.
Also display the emps to whom dept is not assigned.

ename	sal	dname	loc

emp e   dept d

```
INSERT INTO emp(empno,ename,sal) VALUES(1001,'A',6000);
INSERT INTO emp(empno,ename,sal) VALUES(1002,'B',8000);
COMMIT;
```

ORACLE STYLE:

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

ANSI STYLE:

```
SELECT e.ename, e.sal, d.dname, d.loc
FROM emp e LEFT OUTER JOIN dept d
ON e.deptno=d.deptno;
```

Right Outer Join:

- Right Outer Join = matched + unmatched records from Right table

- Right Outer Join can give matched records and unmatched records from right table.

- In ORACLE STYLE, we write (+) symbol at left side.

- In ANSI STYLE, we use the keyword: RIGHT [OUTER] JOIN

Example on Right Outer join:

Display emp details along with dept details.
Also display the depts which are not having emps:

ename	sal	dname	loc

ORACLE STYLE:

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

ANSI STYLE:

```
SELECT e.ename, e.sal, d.dname, d.loc
FROM emp e RIGHT OUTER JOIN dept d
ON e.deptno=d.deptno;
```

Full Outer join:

- Full outer join = matched + unmatched from left and right tables

- **Full outer join can give matched records, unmatched records from left and right tables.**
- **IN ORACLE STYLE, use UNION operator between Left outer Join and Right outer Join.**

  In ORACLE STYLE

Left Outer Join	**e.deptno=d.deptno(+)**
Right Outer Join	**e.deptno(+)=d.deptno**
Full Outer Join	**e.deptno(+) = d.deptno(+) ERROR**
Full Outer Join	**left outer join UNION right outer join**

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

A={1,2,3,4,5}
B={4,5,6,7,8}

A U B = {1,2,3,4,5,6,7,8}

Left Outer Join **= matched + unmatched from left**
UNION
Right Outer Join **= matched + unmatched from right**

Full outer = matched + unmatched from left + unmatched 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

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 = matched + unmatched records from left table

Left outer Join + Condition = unmatched records from left table

Example on Left outer Join + Condition:

Display the emps to whom dept is not assigned as following:

ename	sal	dname	loc
A	6000		
B	8000		

ORACLE STYLE:

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

ANSI STYLE:

```
SELECT e.ename, e.sal, d.dname, d.loc
```

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

Right Outer join = matched + unmatched records from right table

Right outer Join + Condition = unmatched records from right table

Example on Right outer Join + Condition:

Display the depts which are not having emps as following:

ename	sal	dname	loc
		OPERATION	BOSTON

ORACLE STYLE:

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

ANSI STYLE:

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

Full Outer Join = matched + unmatched from left and right tables

Full Outer Join + Conditions = unmatched from left and right tables

Example on Full Outer Join + Conditions:

Display the emps to whom dept is not assigned.
Also display the depts in which emps are not existed as following:

ename	sal	dname	loc
A	6000		
B	8000		
		OPERATIONS	BOSTON

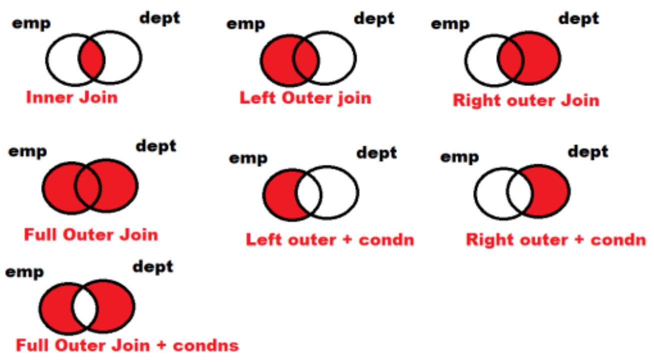
ORACLE STYLE:

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

ANSI STYLE:

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

Venn Diagrams of Joins:



Self Join:

- **If a table is joined to itself then it is called "Self Join".**

- **In Self Join, one table record will be joined with another record in same table.**

Example:

e.mgr = m.empno

EMP e

EMPNO	ENAME	JOB	SAL	MGR
1001	A	MANAGER	30000	
1002	B	CLERK	10000	1001
1003	C	ANALYST	8000	1001
1004	D	MANAGER	25000	
1005	E	SALESMAN	12000	1004
1006	F	CLERK	11000	1004

EMP m

EMPNO	ENAME	JOB	SAL	MGR
1001	A	MANAGER	30000	
1002	B	CLERK	10000	1001
1003	C	ANALYST	8000	1001
1004	D	MANAGER	25000	
1005	E	SALESMAN	12000	1004
1006	F	CLERK	11000	1004

Display emp details with managers details:

emp_name	emp_sal	mgr_name	mgr_sal

ORACLE STYLE:

```
SELECT e.ename AS emp_name, e.sal AS emp_Sal,
m.ename AS mgr_name, m.sal AS mgr_Sal
FROM emp e, emp m
WHERE e.mgr=m.empno;
```

ANSI STYLE:

```
SELECT e.ename AS emp_name, e.sal AS emp_Sal,
m.ename AS mgr_name, m.sal AS mgr_Sal
FROM emp e INNER JOIN emp m
ON e.mgr=m.empno;
```

Display the emp records who are earning more than their managers:

emp_name	emp_Sal	mgr_name	mgr_sal

ORACLE STYLE:

```
SELECT e.ename AS emp_name, e.sal AS emp_sal,
m.ename AS mgr_name, m.sal AS mgr_Sal
FROM emp e, emp m
WHERE e.mgr=m.empno AND e.sal>m.sal;
```

ANSI STYLE:

```
SELECT e.ename AS emp_name, e.sal AS emp_sal,
m.ename AS mgr_name, m.sal AS mgr_Sal
FROM emp e INNER JOIN emp m
ON e.mgr=m.empno
WHERE e.sal>m.sal;
```

Display the emp records who are reporting to BLAKE:

emp_name	mgr_name

ORACLE STYLE:

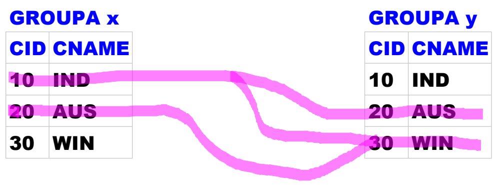
```
SELECT e.ename AS emp_name,
m.ename AS mgr_name
FROM emp e, emp m
WHERE e.mgr=m.empno AND m.ename='BLAKE';
```

ANSI STYLE:

```
SELECT e.ename AS emp_name,
m.ename AS mgr_name
FROM emp e INNER JOIN emp m
ON e.mgr=m.empno
WHERE m.ename='BLAKE';
```

Example:

x.cid < y.cid



GROUPA x

CID	CNAME
10	IND
20	AUS
30	WIN

GROUPA y

CID	CNAME
10	IND
20	AUS
30	WIN

IND VS AUS
IND VS WIN
AUS VS WIN

```
create table groupa
(
cid number(2),
cname varchar2(10)
);

insert into groupa values(10,'IND');
insert into groupa values(20,'AUS');
insert into groupa values(30,'WIN');
commit;
```

ORACLE STYLE:

```
SELECT x.cname || ' VS '|| y.cname
FROM groupA x, groupA y
WHERE x.cid<y.cid;
```

ANSI STYLE:

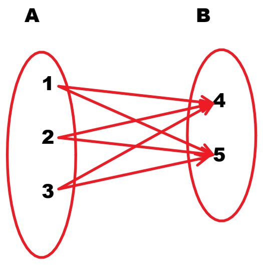
```
SELECT x.cname || ' VS '|| y.cname
```

```
FROM groupA x INNER JOIN groupA y
ON x.cid<y.cid;
```

Cross Join / Cartesian Join:

A = {1,2,3}
B = {4,5}

AXB = ?



AXB = (1,4)(1,5)
(2,4)(2,5)
(3,4)(3,5)

3 * 2 = 6

- in CROSS JOIN, each record in one table will be joined with every record in another table.
- for CROSS JOIN don't write any Join Condition.

Example on CROSS JOIN:

GROUPA a

CID	CNAME
10	IND
20	AUS
30	WIN

GROUPB b

CID	CNAME
40	ENG
50	SL
60	NZ

IND VS ENG
IND VS SL
IND VS NZ
AUS VS ENG
AUS VS SL
AUS VS NZ
WIN VS ENG
WIN VS SL
WIN VS NZ

```
create table groupa
(
cid number(2),
cname varchar2(10)
);

insert into groupa values(10,'IND');
insert into groupa values(20,'AUS');
insert into groupa values(30,'WIN');
commit;

create table groupb
(
cid number(2),
cname varchar2(10)
);

insert into groupb values(40,'ENG');
insert into groupb values(50,'SL');
insert into groupb values(60,'NZ');
commit;
```

ORACLE STYLE:

```
SELECT a.cname || ' VS ' || b.cname
FROM groupA a, groupB b;
```

ANSI STYLE:

```
SELECT a.cname || ' VS ' || b.cname
FROM groupA a CROSS JOIN groupB b;
```

JOINS:
GOAL:
it is used to retrieve the data from multiple tables.

Types of joins:

INNER JOIN		matched records only
	EQUI	based on =, join operation will be performed Example: WHERE e.deptno = d.deptno
	NON-EQUI	based on other than =, join operation will be performed Example: WHERE e.deptno > d.deptno
OUTER JOIN		matched + unmatched records
	LEFT OUTER	matched + unmatched from left
	RIGHT OUTER	matched + unmatched from right
	FULL OUTER	matched + unmatched from left and right
SELF JOIN		a table will be joined to itself
CROSS		each record in 1 table will be joined with every record in another

EMP

EMPNO	ENAME	SAL	DEPTNO
7369	SMITH	800	20
7499	ALLEN	1600	30
7521	WARD	1250	30
7566	JONES	2975	20
7782	CLARK	2450	10
7934	MILLER	1300	10
1001	A	1800	
1002	B	2000	

DEPT

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

Assignment

Wednesday, May 29, 2024 7:18 PM

Assignment:

EMPLOYEE e

EMPID	ENAME	PID
1001	A	30
1002	B	30
1003	C	10
1004	D	10
1005	E	
1006	F	

PROJECT p

PID	PNAME
10	X
20	Y
30	Z

Display emp details along with project details => [Equi Join]

empid	ename	pname

Display emp details along with project details
Also display the employees who are on bench [emps who are not participating in any project development]=> [Left Outer Join]

empid	ename	pname

Display emp details along with project details
Also display the projects which are not assigned to any employee => [Right Outer Join]

empid	ename	pname

display the employees who are on bench

empid	ename	pname

Left outer join + condition

display the projects which are not assigned to any employee

empid	ename	pname

Right outer join + condition

display the employees who are on bench.
display the projects which are not assigned to any employee

empid	ename	pname

full outer join + conditions

Display emp details along with project details.
also display the employees who are on bench.
also display the projects which are not assigned to any employee

empid	ename	pname

full outer join

SUB QUERIES

Wednesday, May 29, 2024 7:23 PM

SYNTAX:

```
SELECT <columns_list> / *
FROM <table_name>
WHERE <column> <operator> (<SELECT QUERY>);
```

Outer / Main / Parent Query **Inner / Sub / Child Query**

Sub Queries / Nested Queries:

- **A query which is written in another query is called "Sub Query".**
- **Outside query is called "Outer / Main / Parent Query".**
- **Inside query is called "Inner / Sub / Child Query".**
- **When we don't know filter condition value to find it we write sub query.**
- **Inner Query must be SELECT only. Inner query cannot be INSERT / UPDATE / DELETE. Because, Inner query has to find some value. Only SELECT can find the value.**
- **Outer query can be INSERT / UPDATE / DELETE / SELECT.**
- **Sub Query must be written in parenthesis.**
- **First INNER QUE€RY gets executed. Then OUTER QUERY gets executed. The result of INNER QUERY will become input for OUTER QUERY.**
- **In WHERE clause we can write max of 254 Sub Queries.**

Types of Sub Queries:

2 Types:

- **Non-Correlated Sub Query**
    - **Single Row Sub Query**
    - **Multi Row Sub Query**
    - **Inline View / Inline Sub Query**
    - **Scalar Sub Query**
- **Correlated Sub Query**

Non-Correlated Sub Query:

- **In Non-Correlated Sub Query,**
  First INNER QUERY gets executed. Then OUTER QUERY gets executed.

- **This 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:

- **If Sub query returns 1 row then it is called "Single Row Sub Query".**

Examples:

Display the emp records who are earning more than BLAKE:

```
SELECT ename, sal
FROM emp
WHERE sal>(find BLAKE sal);

find BLAKE sal:
SELECT sal FROM emp WHERE ename='BLAKE';
SELECT ename, sal
FROM emp
WHERE sal>(SELECT sal FROM emp
WHERE ename='BLAKE');
```

Display the emp records whose job title is same as SMITH:

```
SELECT ename, job, sal
FROM emp
WHERE job=(find SMITH job title);
```

```
find SMITH job title:
SELECT job FROM emp WHERE ename='SMITH';
```

```
SELECT ename, job, sal
FROM emp
WHERE job=(SELECT job FROM emp
WHERE ename='SMITH');
```

Find max salary:

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

Find 2nd max salary:

```
SAL
--------
5000
8000
6000
3000
2000
```

```
SELECT max(sal) FROM emp
WHERE sal<(find max sal);
```

```
5000
6000    ->    max sal => 6000
3000          2nd max sal
2000
```

```
SELECT max(Sal) FROM emp
WHERE sal<(SELECT max(Sal) FROM emp);
```

Find 3rd max sal:

```
SAL
--------
5000
8000
6000
3000
2000
```

```
SELECT max(sal) FROM emp
WHERE sal<(find 2nd max sal);
```

```
5000    ->    max sal  =>  5000
3000          3rd max sal
2000
```

```
SELECT max(Sal) FROM emp
WHERE sal<(SELECT max(Sal) FROM emp
WHERE sal<(SELECT max(Sal) FROM emp));
```

Find the emp who is earning max sal:

```
SELECT ename, max(Sal) FROM emp;
```

Output:
ERROR

When we use Aggregate Function [group function],
SELECT clause allows group function or group by column only.

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

Output:
displays min sal and max sal

```
SELECT deptno, max(Sal) FROM emp
GROUP BY deptno;
```

Output:
displays dept wise max sal

```
SELECT deptno, min(sal), max(Sal) FROM emp
GROUP BY deptno;
--displays dept wise max sal and min sal
```

```
SELECT deptno, ename, max(Sal) FROM emp
GROUP BY deptno;
Output:
ERROR
```

Note:
When we use Aggregate Function [group function],
SELECT clause allows group function or group by
column only.

When we use GROUP BY,
SELECT clause allows group function or group by column only.

Find the emp who is earning max sal:

```
SELECT ename FROM emp
WHERE sal=(find max sal);

SELECT ename FROM emp
WHERE sal=(SELECT max(Sal) FROM emp);
```

Find the emp name who is earnign 2nd max sal:

```
SELECT ename FROM emp
WHERE sal=(find 2nd max sal);

SELECT ename FROM emp
WHERE sal=(SELECT max(sal) FROM emp
WHERE sal<(SELECT max(Sal) FROM emp));
```

Display most seniors record:

```
SELECT ename, sal, hiredate
FROM emp
WHERE hiredate=(find most senior's hiredate);
```

```
SELECT ename, sal, hiredate
FROM emp
WHERE hiredate=(SELECT min(hiredate) FROM emp);
```

Display most juniors record:

```
SELECT ename, sal, hiredate
FROM emp
WHERE hiredate=(find most junior's hiredate);

SELECT ename, sal, hiredate
FROM emp
WHERE hiredate=(SELECT max(hiredate) FROM emp);
```

update JAMES salary as deptno 30's max sal:

```
UPDATE emp
SET sal=(find deptno 30's max sal)
WHERE ename='JAMES';

find deptno 30's max sal:
SELECT max(Sal) FROM emp WHERE deptno=30;


UPDATE emp
SET sal=(SELECT max(sal) FROM emp WHERE deptno=30)
WHERE ename='JAMES';
```

Delete most senior's record:

```
DELETE FROM emp
WHERE hiredate=(find most senior's hiredate);

DELETE FROM emp
WHERE hiredate=(SELECT min(hiredate) FROM emp);
```

Find the deptno which is spending max amount on their emps:

```
SELECT deptno FROM emp
GROUP BY deptno
HAVING sum(sal)=(find max amount in dept wise sum of salaries);
```

```
SELECT deptno FROM emp
GROUP BY deptno
HAVING sum(sal)=(SELECT max(sum(Sal)) FROM emp
GROUP BY deptno);
```

Find the dept name which is spending max amount on their emps:

```
SELECT dname FROM dept
WHERE deptno=(find the deptno which sis spending max amount);
```

```
SELECT dname FROM dept
WHERE deptno=(SELECT deptno FROM emp
GROUP BY deptno
HAVING sum(Sal)=(SELECT max(sum(Sal)) FROM emp
GROUP BY deptno));
```

Multi Row Sub Query:

- **if sub query returns multiple rows then it is called "Multi Row Sub Query".**

- **In this we use IN, ANY, ALL operators.**

Examples on multi row sub query:

Display the emp records whose job titles are same as JAMES and BLAKE job titles [display clerks, managers]:

```
SELECT ename, job, sal
FROM emp
WHERE job=(find JAMES and BLAKE job titles);
```

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

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

ALL:

- **it is used to compare column value with multiple values.**

Syntax:

<column> <relational_operator> ALL(<values_list>)

Example:

Display the emp records whose salary is more than 2000 and 3000:

```
SELECT ename, sal
FROM emp
WHERE sal>ALL(2000,3000);
```

sal>ALL(2000,3000) if sal > all list of values then condition is TRUE SAL --------- 5000 T 4000 T 2500 F 1000 F 1800 F	sal>2000 AND sal>3000
sal<ALL(2000,3000) SAL --------- 5000 F 4000 F 2500 F 1000 T 1800 T	sal<2000 AND sal<3000

ANY:

- **it is used to compare column value with multiple values.**

Syntax:

```
<column> <relational_operator> ANY(<values_list>)
```

Example:

Display the emp records whose salary is more than 2000 or 3000:

```
SELECT ename, sal
FROM emp
WHERE sal>ANY(2000,3000);
```

sal>ANY(2000,3000) **if sal > any one of list of values then condition is TRUE** **SAL** **---------** **5000 T** **4000 T** **2500 T** **1000 F** **1800 F**	**sal>2000 OR sal>3000**
sal<ANY(2000,3000) **SAL** **---------** **5000 F** **4000 F** **2500 T** **1000 T** **1800 T**	**sal<2000 OR sal<3000**

sal=2000 OR sal=3000	**sal IN(2000,3000)**	**sal=ANY(2000,3000)**

Display the emp records who are earning more than all managers:

```
SELECT ename, sal
FROM emp
WHERE sal>ALL(find all managers salaries);
```

```
SELECT ename, sal
FROM emp
WHERE sal>ALL(SELECT sal FROM emp
WHERE job='MANAGER');
```

(or)

```
SELECT ename, sal
FROM emp
WHERE sal>(find max sal in all managers);
```

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

Display the emp records who are earning more than any one of managers:

```
SELECT ename, sal
FROM emp
WHERE sal>ANY(find all managers salaries);
```

```
SELECT ename, sal
FROM emp
WHERE sal>ANY(SELECT sal FROM emp
WHERE job='MANAGER');
```

(or)

```
SELECT ename, sal
FROM emp
WHERE sal>(find min sal in all managers);
```

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

Inline View:

Syntax:

```
SELECT <columns_list>
FROM (<SUB QUERY>)
WHERE <condition>;
```

- **If sub query is written in FROM clause then it is called "Inline View".**
- **Sub Query acts like table.**
- **To control the execution order of clauses we need to write SUB QUERY in FROM clause.**

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
```

Execution Order:

FROM
WHERE
GROUP BY
HAVING
SELECT
DISTINCT
ORDER BY
OFFSET
FETCH

```
SELECT DISTINCT sal
FROM (SELECT ename, sal,
dense_rank() OVER(ORDER By sal DESC) AS rank
FROM emp)
WHERE rank=3;
```

Find 5th max salary:

```
    SELECT DISTINCT sal
    FROM (SELECT ename, sal,
    dense_rank() OVER(ORDER By sal
    DESC) AS rank
    FROM emp)
    WHERE rank=5;
```

Find 10th max sal:

```
SELECT DISTINCT sal
FROM (SELECT ename, sal,
dense_rank() OVER(ORDER By sal DESC)
AS rank
FROM emp)
WHERE rank=10;
```

Find nth max sal:

```
SELECT DISTINCT sal
FROM (SELECT ename, sal,
dense_rank() OVER(ORDER By sal DESC) AS rank
FROM emp)
WHERE rank=&n;

Output:
enter ... n: 3
gives 3rd max sal

/
enter ... n: 5
gives 5th max sal
```

Find top 3 salaried emp records:

```
  SELECT ename, sal,
  dense_rank() over(order by sal desc) as rank
  FROM emp
  WHERE rank<=3;
```

Output:

ERROR: RANK invalid identifier

```
SELECT *
FROM (SELECT ename, sal,
dense_rank() over(order by sal desc) as rank
FROM emp)
WHERE rank<=3;
```

*	All columns of sub query

Find top 5 salaried emp records:

```
SELECT *
FROM (SELECT ename, sal,
dense_rank() over(order by sal desc) as rank
FROM emp)
WHERE rank<=5;
```

Find top n salaried emp records:

```
SELECT *
FROM (SELECT ename, sal,
dense_rank() over(order by sal desc) as rank
FROM emp)
WHERE rank<=&n;
```

Pseudo Columns: **Pseudo => false**

- **ROWNUM**

ROWNUM:

- **ROWNUM is a pseudo column.**

- **It is used to apply row numbers to records.**
- **row number will be applied on result of select query.**

Examples on ROWNUM:

apply row numbers to all emp records:

```
SELECT rownum as sno, empno, ename, sal
FROM emp;
```

apply row numbers to all managers records:

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

Display 3rd row from emp table:

```
SELECT *
FROM (SELECT rownum as rn, empno, ename, sal
FROM emp)
WHERE rn=3;
```

Display 3rd, 7th and 11th rows from emp table:

```
SELECT *
FROM (SELECT rownum as rn, empno, ename, sal
FROM emp)
WHERE rn IN(3,7,11);
```

Display 5th row to 10th row:

```
SELECT *
FROM (SELECT rownum as rn, empno, ename, sal
FROM emp)
WHERE rn BETWEEN 5 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 sub query is written in SELECT clause then it is called "Scalar Sub Query".**
- **It acts like column.**

Examples on Scalar Sub Query:

Display no of records in emp and dept tables:

```
SELECT (SELECT count(*) FROM emp) AS emp,
(SELECT count(*) FROM dept) AS dept
FROM dual;
```

Output:

```
EMP        DEPT
---------- ------------
13          4
```

Calculate share of each dept:

DEPTNO	SUM_OF_SAL	AMOUNT	PER
10	10000	30000	10000*100/30000 = 33.3333
20	?		
30	?		

```
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:

- **First inner query gets executed.**
- **inner query gets executed only once.**

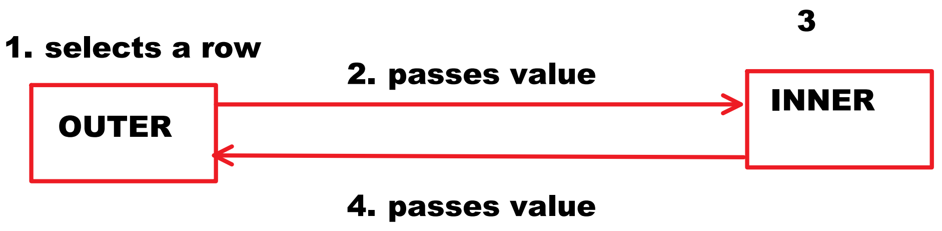
Correlated Sub Query:

- **If outer query passes value to inner query then it is called "Correlated Sub Query".**

- **In Correlated sub query first outer query gets executed. Then inner query gets executed.**

- **Inner query gets executed for multiple times.**

Execution process of Correlated Sub Query:



5. condition => T selects row

1. **Outer query gets executes. It selects a row.**
2. **Outer query passes value to Inner query.**
3. **Inner query gets executed.**
4. **Inner query passes value to Outer query**
5. **Outer query condition will be tested. If condition is TRUE, selects the row.**

Above 5 steps will be executed repeatedly for every row selected by OUTER QUERY.

Example:

Display the emp records who are earning more than their dept's avrg sal:

EMP e

dept's avrg sal:

EMP e

EMPNO	ENAME	DEPTNO	SAL
1001	A	10	20000
1002	B	20	10000
1003	C	30	15000
1004	D	10	10000
1005	E	20	40000
1006	F	30	5000

DEPTNO	AVG(SAL)
10	15000
20	25000
30	10000

Display the emp records who are earning more than their dept's avrg sal:

WHERE sal>(find emp dept's avg sal)

```
SELECT ename, deptno, sal
FROM emp e
WHERE sal>(SELECT avg(Sal) FROM emp
WHERE deptno=e.deptno);
```

Output:

ENAME	DEPTNO	SAL
A	10	20000
C	30	15000
E	20	40000

Display the emp records who are earning max salary in their dept:

EMP e

EMPNO	ENAME	DEPTNO	SAL
1001	A	10	20000
1002	B	20	10000
1003	C	30	15000
1004	D	10	10000
1005	E	20	40000
1006	F	30	5000

WHERE sal = (emp dept's max sal)

```
SELECT ename, deptno, sal
FROM emp e
WHERE sal = (SELECT max(Sal) FROM emp
WHERE deptno=e.deptno);
```

ENAME	DEPTNO	SAL
A	10	20000
C	30	15000
E	20	40000

Sub Query:
A query which is written in another query

Types of Sub Queries:

Non-Correlated		first inner query gets executed inner query gets executed 1 time
	Single Row S Q	SQ returns 1 row
	Multi Row S Q	SQ returns multiple rows
	Inline View	writing SQ in FROM clause
	Scalar S Q	writing SQ in SELECT clause
Correlated		first outer query gets executed inner query gets executed multiple times

Display the emp records who are earning more than BLAKE:

```
SELECT ename, sal
FROM emp
WHERE sal>(find BLAKE sal);

find BLAKE sal:
SELECT sal FROM emp WHERE ename='BLAKE';

SELECT ename, sal
FROM emp
WHERE sal>(SELECT sal FROM emp
WHERE ename='BLAKE');
```

ROWID

Saturday, June 1, 2024 7:36 PM

ROWID:

- **it is a pseudo column.**
- **it is used to get address of row.**

Display address of all emp table records:

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

SELECT rowid, e.*
FROM emp e;
```

Example:

STUDENT

SID	SNAME	SCITY
1001	A	HYD
1001	A	HYD

AAAStWAAHAAAAF7AAB
AAAStWAAHAAAAF7AAC

delete duplicate record:

```
DELETE FROM student
WHERE rowid='AAAStWAAHAAAAF7AAC';
```

CONSTRAINTS

Monday, June 3, 2024 6:13 PM

Constraint:

- **Constraint => restrict / control / limit**
- **Constraint is a rule that is applied on column.**
- **Constraint is used to restrict the user from entering invalid data.**
- **Constraint is used to maintain quality and accurate data.**
- **Maintaining quality data and accurate data is called "Data Integrity".**
- **To implement data integrity feature we use CONSTRAINTS.**

Examples:

Max marks: 100
0 TO 100

STUDENT

CHECK(m1 BETWEEN 0 AND 100)

SID	SNAME	M1
1234	A	78
1235	B	66
1236	C	567 ERROR

CHECK(gender IN('M','F'))

```
GENDER
-------------
M
F
F
M
Z   ERROR
```

ORACLE SQL provides following Constraints:

- **Primary key**
- **Not null**
- **Unique**
- **Check**
- **Default**
- **References / Foreign Key**

Primary key:

- it does not accept duplicates
- it does not accept nulls
- When value is mandatory and it should not be duplicated then use PRIMARY KEY.
- A table can have one primary key only.

Example:

EMPLOYEE

PK

EMPNO		ENAME	JOB	SAL
1001		SAI	CLERK	8000
1002		KIRAN	CLERK	7000
1003		SAI	SALESMAN	8000
1001	ERROR: duplicate	AMAR	MANAGER	15000
	ERROR: null	RAMESH	ANALYST	6000

Example:

```
CREATE TABLE t1
(
f1 INT PRIMARY KEY
);

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

INSERT INTO t1 VALUES(2);
--ERROR: unique constraint violated

INSERT INTO t1 VALUES(null);
--ERROR: cannot insert NULL INTO c##batch6pm.T1.F1
```

NOT NULL:

- it does not accept nulls.
- it accepts duplicates.
- When value is mandatory and it can be duplicated then use NOT

NULL.

Example:

EMPLOYEE

NOT NULL

EMPNO	ENAME	SAL
1234	Raju	8000
1235	Kiran	10000
1236	Raju	6000
1237	ERROR: null	9000

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

CID	CNAME	MOBILE
1234	A	9123456789
1235	B	8976543211
1236	C	

1237	D	9123456789 ERROR
1238	E	

Example:

```
CREATE TABLE t3
(
f1 INT UNIQUE
);

INSERT INTO t3 VALUES(1);
INSERT INTO t3 VALUES(1);   --ERROR
INSERT INTO t3 VALUES(null);
```

CONSTRAINT	DUPLICATE	NULL
PRIMARY KEY	NO	NO
NOT NULL	YES	NO
UNIQUE	NO	YES

PRIMARY KEY = UNIQUE + NOT NULL

CHECK:

- It is used to apply our own condition on column.

Max marks: 100
0 TO 100

Example:

STUDENT

CHECK(m1 BETWEEN 0 AND 100)

SID	SNAME	M1
101	A	77
102	B	56
103	C	678 ERROR

DEFAULT:

- **It is used to apply default value to column.**

Example:

STUDENT

		DEFAULT 'NARESH'	DEFAULT 'HYD'	DEFAULT 20000
SID	**SNAME**	**CNAME**	**CCITY**	**FEE**
1234	A	NARESH	HYD	20000
1235	B	NARESH	HYD	20000
1236	C	NARESH	HYD	20000
1237	D	NARESH	HYD	20000
1238	E	NARESH	HYD	10000

REFERENCES / FOREIGN KEY:

- **Foreign Key can accept PK values of another table.**
- **FK can accept duplicates and nulls.**

Example:

COURSE

PK	
CID	**CNAME**
10	JAVA
20	PYTHON
30	HTML

STUDENT

		FK REFERENCES COURSE(CID)
SID	**SNAME**	**CID**
1001	A	10 ✓
1002	B	10
1003	C	30 ✓
1004	D	20 ✓
1005	E	80 ERROR
1006	F	90 ERROR
1007	G	

Example:

EMPLOYEE

EMPID	ENAME	GENDER	SAL

EMPID	**don't accept dups and nulls**	**PK**
ENAME	**should not be null**	**NOT NULL**
gender	**M or F**	**CHECK**
SAL	**should not be less than 5000**	**CHECK**

```
CREATE TABLE employee
(
empid NUMBER(4) PRIMARY KEY,
ename VARCHAr2(10) NOT NULL,
gender CHAR CHECK(gender IN('M','F')),
sal NUMBER(8,2) CHECK(sal>=5000)
);
```

Example:

STUDENT

SID	SNAME	M1

SID	**don’t accept dups and nulls**
sname	**shoud not be null**
m1	**between 0 and 100**

```
CREATE TABLE student
(
sid NUMBER(4) PRIMARY KEY,
sname VARCHAR2(10) NOT NULL,
m1 NUMBER(3) CHECK(m1 BETWEEN 0 AND 100)
);
```

Example:

users_list

userid	uname	pwd

user_id	don't accept duplicates and nulls	PK
uname	don't accept duplicates and nulls	UNIQUE NOT NULL
pwd	min 8 chars	CHECK

```
CREATE TABLE users_list
(
user_id NUMBER(4) PRIMARY KEY,
uname VARCHAR2(20) UNIQUE NOT NULL,
pwd VARCHAR2(30) CHECK(length(pwd)>=8)
);
```

Example:

STUDENT18

SID	SNAME	CNAME	CCITY	FEE

SID	PK
SNAME	NOT NULL
CNAME	DEFAULT 'NARESH'
CCITY	DEFAULT 'HYD'
FEE	DEFAULT 20000

```
CREATE TABLE student18
(
sid NUMBER(4) PRIMARY KEY,
sname VARCHAR2(10) NOT NULL,
cname VARCHAR2(6) DEFAULT 'NARESH',
ccity VARCHAR2(3) DEFAULT 'HYD',
fee NUMBER(7,2) DEFAULT 20000
);

INSERT INTO student18(sid, sname) VALUES(1001,'A');
```

Example:

DEPT1

PK

DEPTNO	DNAME
10	HR
20	SALES
30	ACCOUNTS

EMP1

FK

PK REFERENCES dept1(deptno)

EMPNO	ENAME	DEPTNO
1001	A	20
1002	B	20
1003	C	10
1004	D	90 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 and FK columns data types must be same

```
INSERT INTO dept1 VALUES(10,'HR');
INSERT INTO dept1 VALUES(20,'SALES');
INSERT INTO dept1 VALUES(30,'ACCOUNTS');
COMMIT;

INSERT INTO emp1 VALUES(1001,'A',30);
INSERT INTO emp1 VALUES(1002,'B',30);
INSERT INTO emp1 VALUES(1003,'C',null);
INSERT INTO emp1 VALUES(1004,'D',90);    --ERROR
```

Naming Constraints

Monday, June 3, 2024 6:36 PM

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:

- **to identify a constraint uniquely in DB a name is required.**
- **as a developer when we define constraint we have to give constraint name. If we don't give constraint name implicitly ORACLE defines a constraint name.**
  Example: SYS_C123456
- **to disable or enable or to drop the constraints this name is useful.**

user_constraints:

- **it is a system table / readymade table**
- **it maintains all constraints information**

To see constraints information:

```
SELECT constraint_name, constraint_type, table_name
FROM user_constraints;
```

Example:

STUDENT19

SID	SNAME	M1

PK   CHECK
c1   c2

```
CREATE TABLE student19
(
sid NUMBER(4) CONSTRAINT c1 PRIMARY KEY,
sname VARCHAR2(10),
m1 NUMBER(3) CONSTRAINT c2 CHECK(m1 BETWEEN 0 AND 100)
);
```

Note:
we cannot give constraint name to DEFAULT

We can apply constraint at 2 levels. they are:
- **column level**
- **table level**

column level constraint:
- **if constraint is defined in column definition then it is called "column level constraint".**
- **All 6 constraints can be applied at column level.**

table level constraint:
- **if constraint is defined after defining all columns then it is called "table level constraint".**
- **we can apply 4 constraints only at table level.**
  PRIMARY KEY, UNIQUE, CHECK, REFERENCES

Example on table level constraint:

STUDENT21

SID	SNAME	M1

PK **CHECK**
c3 **c4**

```
CREATE TABLE student21
(
sid NUMBER(4),
sname VARCHAR2(10),
m1 NUMBER(3),
CONSTRAINT c3 PRIMARY KEY(sid),
CONSTRAINT c4 CHECK(m1 BETWEEN 0 AND 100)
);
```

Example:

COURSE30
PK c5

CID	CNAME
10	JAVA
20	HTML
30	C#

STUDENT30
FK c6
references course30(cid)

SID	SNAME	CID
1001	A	20
1002	B	30
1003	C	70 ERROR

```
CREATE TABLE course30
(
cid NUMBER(2),
cname VARCHAR2(10),
CONSTRAINT c5 PRIMARY KEY(cid)
);
```

```
CREATE TABLE student30
(
sid NUMBER(4),
sname VARCHAR2(10),
cid NUMBER(2),
CONSTRAINT c6 FOREIGN KEY(cid) REFERENCES course30(cid)
);
```

why table level?

2 reasons:

- to apply combination of columns as PK or UNIQUE
- to use another column name in constraint

Composite Primary Key:
If PK is applied on combination of columns then it is called "Composite Primary Key".

Example:
apply combination of columns as PK:

STUDENT31
PK(SID,SUBJECT)

SID	SNAME	SUBJECT	MARKS
1001	A	M1	70
1001	A	M2	80
1001	A	M3	70
1002	B	M1	66
1002	B	M2	59
1002	B	M3	80
1001		M1 ERROR	
null ERROR			
		null ERROR	

```
CREATE TABLE student31
(
sid NUMBER(4),
sname VARCHAR2(10),
subject CHAR(2),
marks NUMBER(3),
CONSTRAINT c11 PRIMARY KEY(sid,subject)
);
```

Example:

PRODUCTS **CHECK(expiry_date>manufactured_date)**

PID	PNAME	MANUFACTURED_DATE	EXPIRY_DATE
1001	ABC	5-JUN-24	25-DEC-22 ERROR

```
CREATE TABLE products
(
pid NUMBER(4),
pname VARCHAR2(10),
manufactured_date DATE,
expiry_date DATE CONSTRAINT c12 CHECK(expiry_date>manufactured_date)
);
Output:
ERROR: Column check constraint cannot reference other columns
```

```
CREATE TABLE products
(
pid NUMBER(4),
pname VARCHAR2(10),
manufactured_date DATE,
expiry_date DATE,
CONSTRAINT c12 CHECK(expiry_date>manufactured_date)
);
```

ALTERING CONSTRAINTS

Wednesday, June 5, 2024 6:35 PM

ALTER:

Using ALTER command we can,

- **add the columns**
- **drop the columns**
- **rename the columns**
- **modify the data types**
- **modify the field sizes**

- **add the constraints**
- **rename the constraints**
- **disable the constraints**
- **enable the constraints**
- **drop the constraints**

Syntax:

```
ALTER TABLE <table_name> [ADD CONSTRAINT <con_name> <con_type>(<column>)]
                         [RENAME CONSTRAINT <old_name> TO <new_name>]
                         [DISABLE CONSTRAINT <con_name>]
                         [ENABLE CONSTRAINT <con_name>]
                         [DROP CONSTRAINT <con_name>];
```

Example:

STUDENT32

SID	SNAME	M1

```
CREATE TABLE student32
(
sid NUMBER(4),
sname VARCHAR2(10),
m1 NUMBER(3)
);
```

Note:

- **we can add the constraint using ADD CONSTRAINT or MODIFY**
- **using ADD CONSTRAINT, we can add table level constraints only.**

- **using MODIFY, all 6 constraints can be added.**

Add PK to sid:

```
ALTER TABLE student32
ADD CONSTRAINT c20 PRIMARY KEY(sid);
(or)
ALTER TABLE student32
MODIFY sid CONSTRAINT c20 PRIMARY KEY;
```

Add not null to sname:

```
ALTER TABLE student32
MODIFY sname CONSTRAINT c21 NOT NULL;
```

Adding CHECK constraint to m1:

```
    ALTER TABLE student32
    ADD CONSTRAINT c22 CHECK(m1 BETWEEN 0 AND 100);
```

Renaming PK [rename c20 to Z]:

```
ALTER TABLE student32
RENAME CONSTRAINT c20 TO z;
```

Disabling PK:

```
ALTER TABLE student32
DISABLE CONSTRAINT z;
```

Enabling PK:

```
ALTER TABLE student32
ENABLE CONSTRAINT z;
```

Dropping PK:

```
ALTER TABLE student32
DROP COSNTRAINT z;
```

SET OPERATORS

Wednesday, June 5, 2024 7:05 PM

A = {1,2,3,4,5}
B = {4,5,6,7,8}

A U B = {1,2,3,4,5,6,7,8} = B U A
A UA B = {1,2,3,4,5,4,5,6,7,8} = B UA A
A I B = {4,5} = B I A => common elements

A M B = {1,2,3} != B M A=> specific elements of A
B M A = {6,7,8} => specific elements of B

SET OPERATORS:

- **SET OPERATOR is used to combine result of 2 select queries.**

  Syntax:

  ```
  <SELECT QUERY>
         <SET OPERATOR>
  <SELECT QUERY>;
  ```

  ORACLE SQL provides following SET OPERATORS:
  - **UNION**
  - **UNION ALL**
  - **INTERSECT**
  - **MINUS**

  UNION:
  it combines result of 2 select queries without duplicates

UNION ALL:
it combines result of 2 select queries including duplicates

INTERSECT:
it gives common records from the result of 2 select queries.

MINUS:
it gives specific records from select query.

Example:

FOOTBALL

SID	SNAME
1001	A
1002	B
1003	C

CRICKET

SID	SNAME
5001	D
1002	B
5002	E

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

INSERT INTO football VALUES(1001,'A');
INSERT INTO football VALUES(1002,'B');
INSERT INTO football VALUES(1003,'C');
COMMIT;

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

INSERT INTO cricket VALUES(5001,'D');
INSERT INTO cricket VALUES(1002,'B');
INSERT INTO cricket VALUES(5002,'E');
COMMIT;
```

Display the students records who are participating in FOOTBALL and CRICKET:

```
SELECT sid, sname FROM football
UNION
SELECT sid, sname FROM cricket;
```

SID	SNAME
1001	A
1002	B
1003	C
5001	D
5002	E

Display the students records who are participating in FOOTBALL and CRICKET including duplicates:

```
SELECT sid, sname FROM football
UNION ALL
SELECT sid, sname FROM cricket;
```

Output:

SID	SNAME
1001	A
1002	B
1003	C
5001	D
1002	B
5002	E

Display the students who are participating in FOOTBALL and CRICKET:

```
SELECT sid,sname FROM football
INTERSECT
SELECT sid,sname FROM cricket;
```

Display the students who are participating in FOOTBALL and not participating in CRICKET:

```
SELECT sid, sname FROM football
MINUS
SELECT sid, sname FROM cricket;
```

Display the students who are participating in CRICKET and not participating in FOOTBALL:

```
SELECT sid, sname FROM cricket
MINUS
```

```
SELECT sid, sname FROM football;
```

Example:

DEPTNO 10	DEPTNO 20
MANAGER	MANAGER
CLERK	CLERK
PRESIDENT	ANALYST

Display the job titles offered by deptno 10 and 20:

```
SELECT job FROM emp WHERE deptno=10
UNION
SELECT job FROM emp WHERE deptno=20;

MANAGER
CLERK
PRESIDENT
ANALYST
```

Display the common job titles offered by deptno 10 and 20:

```
SELECT job FROM emp WHERE deptno=10
INTERSECT
SELECT job FROM emp WHERE deptno=20;

MANAGER
CLERK
```

Display the specific job titles offered by deptno 10 and not offered by 20:

```
SELECT job FROM emp WHERE deptno=10
MINUS
SELECT job FROM emp WHERE deptno=20;
```

```
PRESIDENT
```

Display the specific job titles offered by deptno 20 and not offered by 10:

```
SELECT job FROM emp WHERE deptno=20
MINUS
SELECT job FROM emp WHERE deptno=10;
```

```
ANALYST
```

Rules in SET OPERATORS:

- **No of columns in both SELECT QUERIES must be same.**

  Example:

  ```
  SELECT sid, sname FROM football
  UNION
  SELECY sid FROM cricket;
  Output:
  ERROR
  ```

- **data types of corresponding columns in both SELECT**

QUERIES must be same.

Example:

```
SELECT sid, sname FROM football
UNION
SELECT sname, sid FROM cricket;
```

Output:

ERROR

Differences b/w UNION and UNION ALL:

UNION	UNION ALL
• it does not give duplicates	• it gives duplicates
• slower	• faster

Differences b/w UNION and JOIN:

UNION	JOIN
• it combines the rows	• it combines the columns
• it is used for horizontal merging	• it is used for vertical merging
• it is applied on similar structures	• it is applied on dissimilar structures

EMP

EMPNO	ENAME	DEPTNO

DEPT

DEPTNO	DNAME

EMPNO	ENAME	DNAME

JOINS

EMP_IND

EMPNO	ENAME
1001	A
1002	B

EMP_US

EMPNO	ENAME
5001	C
5002	D

UNION

EMPNO	ENAME
1001	A
1002	B
5001	C
5002	D

TABLES

Thursday, June 6, 2024 6:38 PM

ORACLE => RDBMS

$$\begin{tikzcd}
\textbf{SQL} \arrow[r, "\textbf{queries}"] & \textbf{DATABASE} \\
\textbf{PL/SQL} \arrow[ur, "\textbf{programs}"'] &
\end{tikzcd}$$

TABLES

ROWS & COLUMNS

SQL sub languages:

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

Built-In Functions:

string	lower() upper() initcap()
	lpad() rpad()
	ltrim() rtrim() trim()
	Substr() Instr()
	Replace() Translate()

conversion	to_char()
	to_date()
	to_number()

aggregate / group	max() min() count()
	avg() sum()

date	add_months()
	sysdate
	systimestamp
	last_day()
	next_day()

analytic	rank() dense_rank()
	row_number()

number	trunc() ceil() floor() round()
other	NVL() NVL2()

Clauses:

FROM
WHERE
GROUP BY
HAVING
SELECT
DISTINCT
ORDER BY
OFFSET
FETCH

Joins:

Goal:
used to retrieve the data from multiple tables

Inner Join	**matched**
equi	**=**
non-equi	**other than =**
Outer Join	**mathced + um**
left outer	**m + um from L**
right outer	**m + um from R**
full outer	**m + um from L & R**
self join	**FROM emp e, emp m**
cross join	

Sub Queries:
a query which is written in another query

Types:

Non-correlated => inner. 1 time
- **single row sq => 1 row**
- **multi row sq => multiple rows**
- **inline view => FROM**
- **scalar sq => SELECT**

Correlated => outer. multiple times

Constraints:

PK
NOT NULL
UNIQUE
CHECK
DEFAULT
FK

SET OPERATORS:

UNION
UNION ALL
INTERSECT
MINUS