

Notes Link:

<https://bit.ly/oracledbnotes>

admin prathyusha :

9121104161 [whatsapp only]

ORACLE installation video link:

<https://bit.ly/orainstall>

note:

install oracle 21c

procedure is same for 19c and 21c

Oracle [SQL & PL/SQL] @ 9:00 AM (IST) By Mr.Shiva Chaitanya

Day-1 <https://youtu.be/HP8OW4R3H6E>

Day-2 <https://youtu.be/0AJKOUVVVMiQ>

Day-3 <https://youtu.be/m67jW6z81h8>

Day-4 <https://youtu.be/-XMfkUe9xHM>

Day-5 <https://youtu.be/aqtCWjGijQc>

Day-6 <https://youtu.be/Q3K-3lelwFM>

Day-7 <https://youtu.be/RkYUyo4wxHc>

Day-8 <https://youtu.be/M65CXIXaXUU>

Day-9 https://youtu.be/_AheqFqBoDE

ORACLE:

Data Store

Database

DBMS

RDBMS

Metadata

Data Store:

The location where data is stored is called "data store".

Examples:

File, Database

Goal:

Storing Business data permanently

int sid;

sid=1001

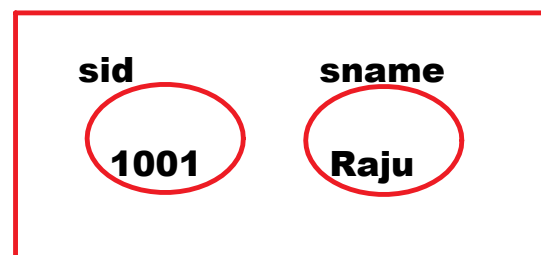
sid => variable

Student s1 = new Student();

sid



s1



- variable and object are temporary.
- to store the data permanently we use **FILE** or **DATABASE**

FILE	DATABASE
<ul style="list-style-type: none"> • developed for single user • used to store small amounts of data • No security 	<ul style="list-style-type: none"> • developed for multiple users • used to store large amounts of data • It is secured

Database:

- Database is a kind of data store.
- Database is a location where organization's business data stored permanently.

Online shopping

Searching for products
Adding to wishlist
Placing order
payment

Amazon DB

Products
Wishlist
Orders
Payments
Customers

Bank DB

**Customers
Transactions
Products
Branches
Staff**

DBMS:

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

Before 1960s => BOOKS

In 1960s => FMS

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

In 1976 => RDBMS [Relational DBMS] => E.F.Codd

ORACLE company Founder => Larry Ellison

In 1979 => ORACLE => RDBMS

RDBMS:

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

Examples:

ORACLE, SQL SERVER, DB2, Postgre SQL, MY SQL

Browser	Laptop
Google Chrome	Dell
Mozilla Firefox	Microsoft
Opera	

Table:

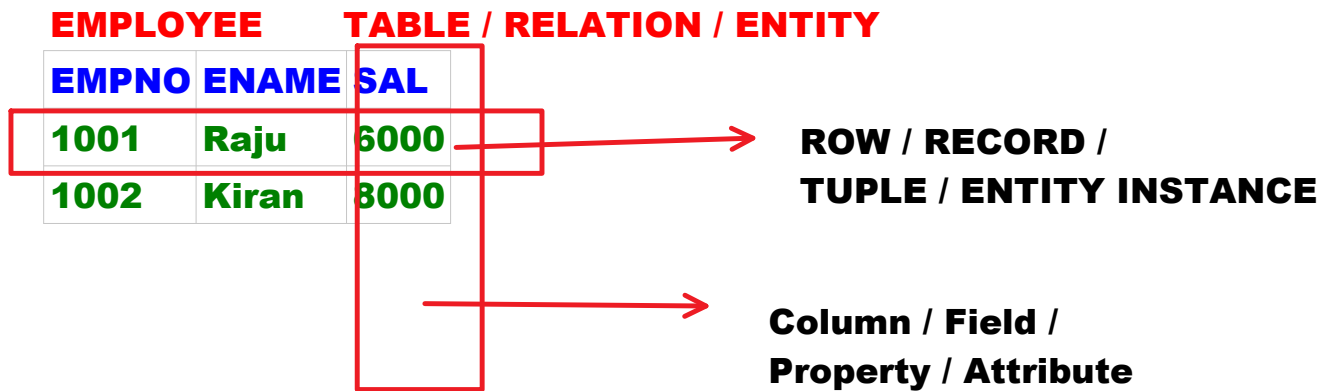
Table is a collection of rows and columns

Row:

*** Horizontal representation of data**

Column:

Vertical representation of data



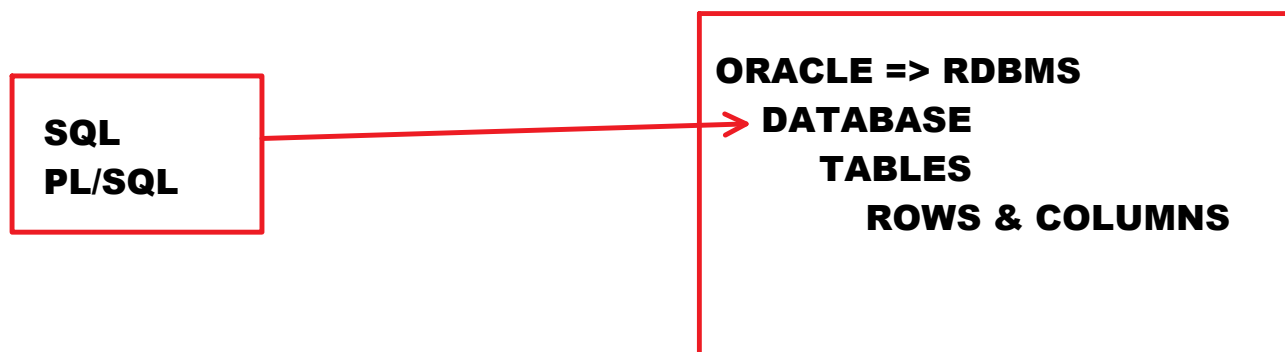
Metadata:

- **Metadata is the data about the data.**
- **Example:**
 - **Field name** => Sid, Sname, Fee
 - **Table name** => STUDENT
 - **Data type** => NUMBER, VARCHAR2, DATE
 - **Field Size** => NUMBER(4), VARCHAR2(10)

STUDENT

Sid NUMBER(4) -9999 TO 9999	Sname VARCHAR2(10)	Fee
1001	Raju	6000
KIRAN ERROR		
25-DEC-2020 ERROR		
1002		
9999		
10000 => ERROR		

Data Store	is a location where data is stored
Database	is a kind of data store is a location where organization's business data stored permanently
DBMS	is a software => used to create and maintain the database
RDBMS	is a software => used to create and maintain the database in the form of tables
Metadata	is the data about the data



Bank DB

Customers TABLE

CID	CNAME	Mobile	MailID	Addhar	PAN
-----	-------	--------	--------	--------	-----

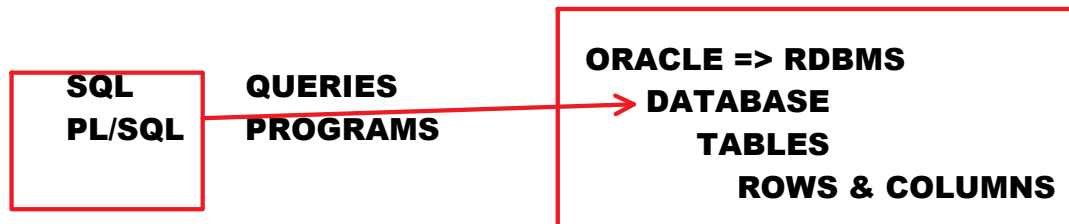
Transactions TABLE

TID	T_DATE_TIME	T_TYPE	AMOUNT
-----	-------------	--------	--------

Branches TABLE

IFSC_CODE	CITY	STATE	COUNTRY
-----------	------	-------	---------

ORACLE



SQL:

TABLES			
	SQL Commands	DDL, DML, DCL, TCL, DRL	
	Built-In Functions	String, Number, Date, Conversion, Analytic,	
	CLAUSES	GROUP BY, HAVING, ORDER BY, ...	
	JOINS	Inner joins, outer joins, cross join ...	
	SUB QUERIES	single row, multi row, correlated, ...	
	SET OPERATORS	UNION, UNION ALL, INTERSECT, ...	
VIEWS	Simple View, Complex View		
INDEXES	B-Tree Index, Bitmap Index		
Materialized Views	refreshing		
Sequences	sequences identity		
Synonyms			

PL/SQL:

PL/SQL Basics	data types declare assign print read using SQL commands
Control Structures	Conditional looping Jumping
CURSORS	steps types of cursors implicit cursor explicit cursor ref cursor parameterized cursor inline cursor
EXCEPTION HANDLING	Built-in Exceptions User-Defined Exceptions Pragma exception_init() raise_application_error()
COLLECTIONS	types: associative array nested table v-array
Stored procedures	
Stored Functions	
Packages	
Triggers	
Working with LOBs	
Dynamic SQL	

ORACLE:

- **ORACLE is a Relational DataBase Management Software.**
- **It is used to create and maintain the database in the form of tables.**
- **database => organization's business data**
- **It allows us to store, manipulate and retrieve the data of database.**

Manipulate => INSERT / UPDATE / DELETE

Examples:

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

Retrieve => opening existing data

Check balance

Searching for products

Transaction Statement

- **ORACLE DB Software 2nd version released in 1979.**
They didn't release 1st version to market.
- **For WINDOWS OS => latest version is: ORACLE 21C**
- **For LINUX OS => latest version is ORACLE 23C**

Before 1960s => BOOKS

1960s => FMS

1970s => HDBMS [heirarchical]
NDBMS [network]

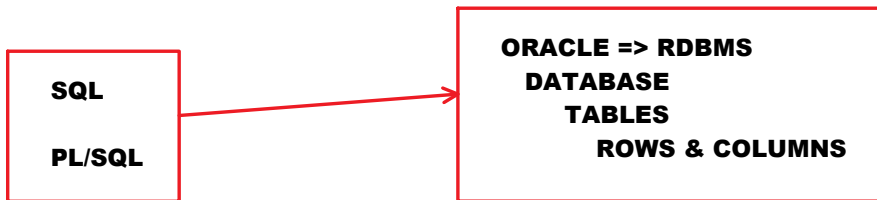
1976 => RDBMS concept => E.F.Codd

Larry Ellison => Founder of ORACLE

1977 => established company
Software Development Laboratories

1979 => renamed => Relational Software Inc.
ORACLE software released

1983 => renamed => ORACLE corp.



To communicate with ORACLE DB, we can use 2 languages.

They are:

- SQL
- PL/SQL

SQL:

- SQL => Structured Query Language
- SQL is a query language.
- SQL is used to write the queries.
- Query => request / command / instruction
- Query is a request that is sent to DB SERVER.
- Queries are written to communicate with ORACLE Database.
- SQL is a Non-Procedural Language. We will not write any set of statements or programs in SQL. Just we write Queries.
- SQL is Unified Language. It is common language to work with many Relational Databases.

In C:

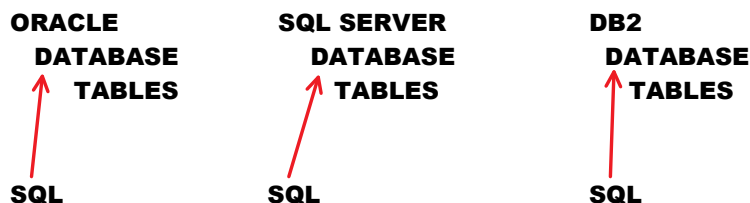
Function => is a set of statements

In Java:

Method => is a set of statements

In PL/SQL:

Procedure => is a set of statements



- SQL is 4GL [4th Generation Language]. 4GLs provide readymade commands and readymade functions.

Find max sal

max(sal)

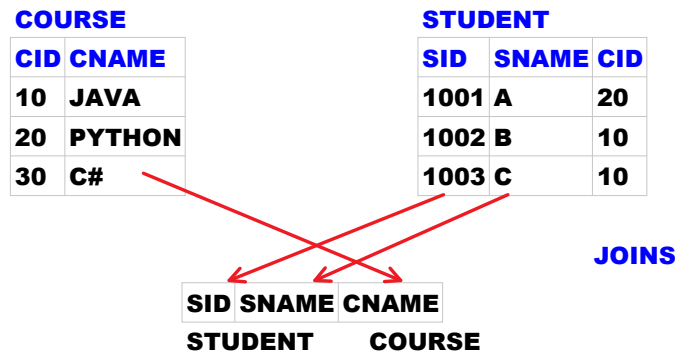
Programming Language

Program => 10 lines code

what to do

how to do

- SQL provides operators to perform operations like arithmetic or logical operations.
- SQL provides JOINS concept to retrieve data from multiple tables



- It provides SUB QUERIES Concept.

Display emp records whose salary is more than BLAKE?

```
SELECT ename,sal FROM emp
WHERE sal>(SELECT sal FROM emp
WHERE ename='BLAKE');
```

PL/SQL:

- PL/ SQL => Procedural Language / Structured Query Language.
- It is programming language.
- It is a procedural language.
procedure => a set of statements / program
- PL/SQL = SQL + Programming
- PL/SQL is extension of SQL.
- In PL/SQL we develop programs to communicate with ORACLE DB.

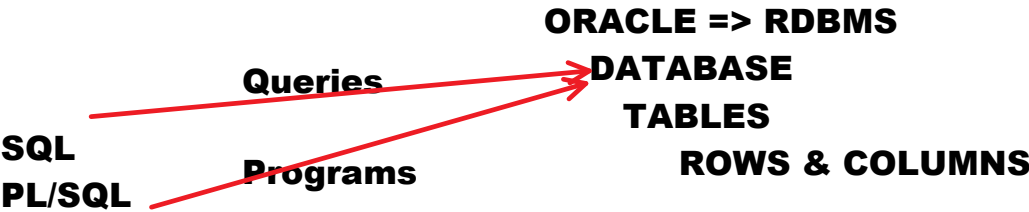
SQL Commands:

SQL provides 5 sub languages. They are:

DDL
DRL / DQL
DML
TCL
DCL / ACL

DDL: <ul style="list-style-type: none">• Data Definition Language• Data Definition => metadata• It deals with metadata	CREATE ALTER DROP FLASHBACK [ORACLE 10g] PURGE [ORACLE 10g] TRUNCATE RENAME
DRL / DQL <ul style="list-style-type: none">• DRL => Data Retrieval Language• DQL => Data Query Language• Retrieve => opening existing data• It deals with Data Retrievals	SELECT
DML <ul style="list-style-type: none">• Data Manipulation Language• manipulation => insert / update / delete• it deals with data manipulations	INSERT UPDATE DELETE INSERT ALL [Oracle 9i] MERGE [Oracle 9i]
TCL <ul style="list-style-type: none">• Transaction Control Language• It deals with transactions.• transaction => is a series of actions	COMMIT ROLLBACK SAVEPOINT
DCL / ACL <ul style="list-style-type: none">• DCL => Data Control Language• ACL => Accessing Control Language	GRANT REVOKE

<ul style="list-style-type: none">• it deals with data accessibility	
--	--



SQL:

DDL [metadata]	DRL / DQL [retrievals]	DML [data]	TCL [transactions]	DCL [accessibility]
create alter drop flashback purge truncate rename	select	insert update delete insert all [oracle 9i] merge [oracle 9i]	commit rollback savepoint	grant revoke

CREATE:

CREATE command is used to create the **ORACLE** DB Objects like Tables, Views, Indexes ...etc.

Bank DB

Customers Table

CID	CNAME	CCITY	AADHAR	PAN	MOBILE
------------	--------------	--------------	---------------	------------	---------------

Transactions Table

TID	T_DATE_TIME	T_TYPE	ACNO	AMOUNT
------------	--------------------	---------------	-------------	---------------

ORACLE DB Objects:

- Tables**
- Views**
- Indexes**
- Materialized Views**
- Sequences**
- Synonyms**
- Stored Procedures**
- Stored Functions**
- Packages**
- Triggers**

Customers Table

CID	CNAME	CCITY	AADHAR	PAN	MOBILE
-----	-------	-------	--------	-----	--------

Transactions Table

TID	T_DATE_TIME	T_TYPE	ACNO	AMOUNT
-----	-------------	--------	------	--------

Synonyms

Stored Procedures

Stored Functions

Packages

Triggers

ALTER:

- **ALTER => Change**
- **ALTER command is used to change structure of the table.**
- **Using ALTER command we can:**
 - **Add the Columns**
 - **Rename the Columns**
 - **Drop the Columns**
 - **Modify the field size**
 - **Modify the data type**

Example:

EMPLOYEE

EMPNO	ENAME	SAL
NUMBER(4)	VARCHAR2(10)	
CHAR(8)		

HYD_1001

DLH_1002

BLR_1003

GENDER	MAIL_ID
---------------	----------------

Note:

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

DROP	used to drop the table when table is dropped, it goes to recyclebin
FLASHBACK	used to restore the dropped table
PURGE	used to delete the table from Recyclebin

Table:

- is a collection of rows and columns.

EMPLOYEE			→ Table
EMPNO	ENAME	SAL	→ Table Structure [columns]
1001	A	6000	
1002	B	7000	→ Table data [rows]

Truncate	used to delete all records from table with good performance
-----------------	---

EMPLOYEE		
EMPNO	ENAME	SAL
1001	A	6000
1002	B	7000

RENAME	used to rename the table
---------------	--------------------------

DML:

Manipulations => Insert / Update /Delete

emp joined	insert emp record	INSERT
emp promoted	update the job title and salary	UPDATE
emp resigned	delete emp record	DELETE

TCL:

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

Examples:

Withdraw, Deposit, Fund Transfer, Placing Order

Fund Transfer:

Accounts

ACNO	NAME	BALANCE
1001	A	100000
1002	B	50000

**transfer 20000 amount from 1001 to 1002:
[Transaction]**

- **sufficient funds? => SELECT**
- **UPDATE from a/c balance => UPDATE**
- **UPDATE to a/c balance => UPDATE**

A transaction must be successfully finished or cancelled.

If transaction is successful	save it	COMMIT
If transaction is unsuccessful	cancel it	ROLLBACK

GRANT => used to give permission to other users

REVOKE => used to cancel the permission.

CREATE:
CREATE command is used to create the tables.

Syntax:

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

EMPLOYEE		
EMPNO	ENAME	SAL

< >	ANY
[]	OPTIONAL

Note:
Till **ORACLE 21C**, we can create max of **1000** columns only.

In **ORACLE 23C**, we can create max of **4096** columns.

Data types in ORACLE SQL:

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

ORACLE SQL provides following data types:

Character Related 'RAJU' 'MANAGER' 'B.Tech'	Char(n) Varchar2(n) LONG CLOB nChar(n) nVarchar2(n) nCLOB
Integer related 1234 78 567 18	NUMBER(p) Integer Int
Floating point related 7000.00 2000.80 67.89	NUMBER(p,s) float binary_float binary_double

Date & time related	DATE TIMESTAMP [oracle 9i]
25-DEC-23 29-FEB-24 10:30.0.0 AM	
Binary related	BFILE BLOB
images, audios, videos	

Character related data types:

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

Char(n):

- it is used to hold string values.
- n => max no of chars.
- Fixed length data type.
- extra memory will be filled with spaces
- max size: 2000 bytes [2000 chars]
- default size: 1

Varchar2(n):

- it is used to hold string values.
- n => max no of chars.
- Variable length data type.
- extra memory will be removed.
- max size: 4000 bytes [4000 chars]
- default size: no default size

Fixed length	T1	F2 VARCHAR2(10)	Variable Length
10	RAMU6SPACES	RAMU	4
10	SAI7spaces	SAI	3
10	NARESH4spaces	NARESH	6

State_Code CHAR(2)

TS
AP
MH
WB
UP

Ename VARCHAR2(10)

Ravi
Kiran
Ramesh
Sai

VEHICLE_NUMBER CHAR(10)	mail_id VARCHAR2(30)
-----	-----
TS08AA1234	sai_lumar1234@gmail.com
	raju@nareshit.com

Note:

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

To hold large amounts of chars we use LONG or CLOB.
CLOB is best one. Because, LONG has some drawbacks.

LONG:

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

CLOB:

- CLOB => Character Large Object
- it is used to hold large amounts of chars.
- We can create multiple columns as CLOB type in a table
- We can use built-in functions on CLOB type column.
- Max size: 4GB

Character Related data types:

Char(n)	• ASCII code Char Data types
Varchar2(n)	• English lang chars only
LONG	• Single Byte Char Data Types
CLOB	
nChar(n)	• UNI code Char Data Types
nVarchar2(n)	• English + other lang chars
nCLOB	• Multi Byte Char Data Types
n => National	

In C: **ASCII**
char ch; // 1 Byte

In Java: **UNI**
char ch; // 2 Bytes

ASCII:

- is a Coding System
- American Standard Code for Information Interchange
- 256 chars are coded.
- ranges from 0 to 255.
- English + digits + special chars
- 255 => 1111 1111 => 8 bits [1 Byte]

UNI:

- is a coding system
- UNI => UNiversal
- 65536 chars are coded
- ranges from 0 to 65535
- It is extension of
- UNI = ASCII + other language chars
- 65535 => 1111 1111 1111 1111 => 16 bits [2 Bytes]

nChar(n)	• It is fixed length data type • n => no of chars • max size: 2000 Bytes [1000 chars]
nVarchar2(n)	• It is variable length data type • n => no of chars • max size: 4000 Bytes [2000 chars]
nCLOB	• To hold more than 2000 chars

use nCLOB

Note:

To hold **ENGLISH** lang chars only use **Normal Char** data types.

To hold other than **ENGLISH** lang chars use **National Char** data types

Integer Related data types:**NUMBER(p):**

- p => precision => max no of digits
- p valid range => 1 to 38
- it is used to hold integers

Examples:**Max marks: 100**

maths_marks **NUMBER(3)** **-999 TO 999**

78

100

45

786

999

1000 => ERROR

EMPID **NUMBER(4)** **-9999 TO 9999**

1001

1002

1003

6789

9999

10000 => ERROR

MobileNumber **NUMBER(10)**

CreditCardNumber **NUMBER(16)**

AadharNumber **NUMBER(12)**

PAN_CARD_NUMBER **CHAR(10)**

ABCDE1234F

Note:

Number related data types can accept digits only

Char related data types can accept letters, digits and special chars.

Note:

Integer and int are alias names of NUMBER(38)

create table t4

```
(  
  f1 number(38),  
  f2 int,  
  f3 integer  
);
```

DESC t4;

Output:

Name	Null?	Type
F1		NUMBER(38)
F2		NUMBER(38)
F3		NUMBER(38)

Floating point related data types:

NUMBER(p,s):

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

Examples:

avg NUMBER(5,2) -999.99 TO 999.99

67.89

45.23

100.00

786.45

999.99

1000 => ERROR

123.45678 => 123.46

123.45378 => 123.45

100.00

p=5

s=2

country_code CHAR(3)

IND

AUS

USA

pname VARCHAR2(10)

laptop

mouse

micro processor

pid NUMBER(6)

123456

123457

SAL NUMBER(8,2)

-9999999.99 TO 9999999.99

SQL => Queries
PL/SQL => Programs

ORACLE => RDBMS
DATABASE
TABLES



SQL => Non Procedural language => Queries

PL/SQL => Procedural Lang => Programs

SQL provides 5 sub languages:

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

CREATE:
used to create the tables

Syntax:

```

CREATE TABLE <name>
(
  <field_name> <data_type>,<
  <field_name> <data_type>,<
  .
  .
);

```

Char(n)	Fixed length chars
Varchar2(n)	Variable length chars 4000 chars only
CLOB	

Customer

cid NUMBER(6)	cname VARCHAR2(10)	gender CHAR(1)	cstate CHAR(2)	ccountry CHAR(3)			
123456	Raju	M	TS	IND			
123457	Naresh	F	AP	AUS			

Aaddhar_Number NUMBER(12)
Mobile_Number NUMBER(10)

maths NUMBER(3)

avrg **NUMBER(5,2)**

100.00

sal **NUMBER(8,2)**

100000.00

height NUMBER(2,1)

5.6

5.4

6.0

Date & Time Related Data Types:

- **Date**
- **Timestamp**

Date:

- **it is used to hold date values.**
- **it can hold date, month, year, hours, minutes and seconds**
- **it cannot hold fractional seconds**
- **default oracle date format: DD-MON-YY [04-MAR-24]**
- **default time: 12:00:00 AM [midnight time]**
- **Fixed length data type.**
- **Max size: 7 Bytes**

Example:

DOJ DATE

DOR DATE

Ordered_Date DATE

Delivery_Date DATE

Transaction_Date DATE

Timestamp:

- **Introduced in ORACLE 9i version.**
- **It is extension of DATE type.**
- **It can hold date, month, year, hours, minutes, seconds and fractional seconds.**

- **It is used to hold date and time values.**
- **default time: 12:00:00.0 AM**
- **it is fixed length data type**
- **max size: 11 Bytes**

Example:

Trans_Date_Time TIMESTAMP

order_date_time TIMESTAMP

manufactured_date_time TIMESTAMP

login_date_time TIMESTAMP

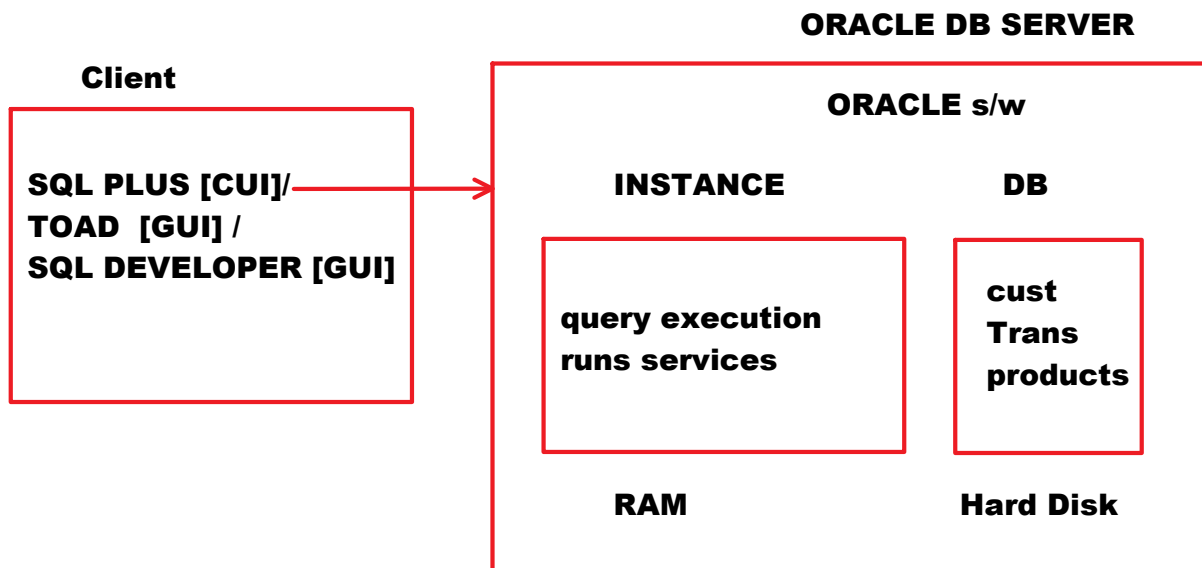
Note:

To hold DATE value use DATE type.

TO hold DATE and TIME value use TIMESTAMP.

DOJ DATE

LOGIN_DATE_TIME TIMESTAMP



ORACLE DB SERVER:

- The machine in which we install **ORACLE DB s/w** that is called "**Oracle DB Server**".
- **DB Server = Instance + DB**
- **DB Server is a combination of INSTANCE and DB.**
INSTANCE will be stored in **RAM**.
DB will be stored in **HARD DISK**.

Note:

- **ORACLE** is **Server side s/w**
- In client machine we install **SQL DEVELOPER** or **TOAD** or **SQL PLUS**

Using Client Side Software we communicate with ORACLE DB [SERVER] by writing QUERIES in SQL language.

Opening SQL PLUS:

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

- Login as DBA:

username: system

password: nareshit

[you have given DBA password at the time of installation. enter that password]

- displays SQL command prompt

SQL> --type the queries

to clear the screen:

Syntax:

CL[EAR] SCR[EEN]

Example:

SQL> CL SCR

Creating User:

- DBA creates the User.

Syntax:

CREATE USER <user_name>
IDENTIFIED BY <password>;

Example on creating user:

create a user with the username "c##batch9am"
and with the password "nareshit":

Login as DBA:

username: system

password: nareshit

CREATE USER c##batch9am

c##batch9am	common user
batch9am	local user

IDENTIFIED BY nareshit;

Output:

User created

**GRANT connect, resource, unlimited tablespace
TO c##batch9am;**

connect	<ul style="list-style-type: none">• is a privilege [permission]• is a permission for login
resource	<ul style="list-style-type: none">• is a permission for creating DB resources like tables
unlimited tablespace	<ul style="list-style-type: none">• is a permission for inserting records

CONN[ECT]:

it is used login from SQL command prompt

Syntax:

CONN[ECT] username/password

Example:

CONN c##batch9am/nareshit

to see current user name:

SQL> SHOW USER

create new user

username: c##userxyz

password: xyz

login as DBA:

**CREATE USER c##userxyz
IDENTIFIED BY xyz;**

Output:

user created

**GRANT connect, resource, unlimited tablespace
TO c##userxyz;**

Output:

GRANT succeeded

Changing password of USER:

Syntax:

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

Example:

change password of the user c##userxyz as nareshit:

login as DBA:

username: system

password: nareshit

**ALTER USER c##userxyz
IDENTIFIED BY nareshit;**

Changing DBA password:

username: sys as sysdba

password: [don't enter any password]

**SQL> ALTER USER system
IDENTIFIED BY naresh;**

Dropping User:

Syntax:

DROP USER <user_name> [CASCADE];

Example:

Login as DBA:

DROP USER c##batch9am CASCADE;

case-1:

USER: c##batch9am



EMPTY

no need to use CASCADE

DROP USER c##batch9am;

USER: c##batch9am

T1

DROP USER c##batch9am CASCADE;

CREATING TABLES

Wednesday, March 6, 2024 9:19 AM

CREATING TABLES & INSERTING RECORDS:

CREATE:

- It is DDL command.
- it is used to create the DB Objects like **tables**, views, indexes ... etc.

Syntax:

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

Till ORACLE 21C	max 1000 columns
In ORACLE 23C	max 4096 columns

INSERT:

- It is DML command.
- It is used to insert the records.

Syntax:

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

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

Examples on creating tables:

Example-1:

STUDENT

SID	SNAME	AVRG
1001	A	67.89
1002	B	56.43

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

to see table structure:

DESC[RIBE]:

this command 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 list of tables created by user:

user_tables:

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

Example:

DESC user_tables

SELECT table_name FROM user_tables;

Output:

TABLE_NAME

STUDENT

Inserting Records:

case-1: inserting single record

case-2: inserting multiple records using parameters

case-3: inserting limited column values

Case-1: inserting single record

1001	A	67.89
-------------	----------	--------------

INSERT INTO student VALUES(1001,'A',67.89);

Output:

1 row created.

1002	B	56.43
------	---	-------

INSERT INTO student VALUES(1002,'B',56.43);

Output:

1 row created.

COMMIT; --saves data in DB

case-2: inserting multiple records using parameters

- **PARAMETER concept is used to insert multiple records.**
- **PARAMETER concept is used to read the values at runtime.**

Syntax:

&<text>

Example:

&sid

Output:

enter value for sid:

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

Output:

enter value for sid: 1003

enter value for sname: C

enter value for avrg: 55.66

INSERT INTO student VALUES(1003,'C',55.66)

1 row created.

/

Output:

enter value for sid: 1004

/ (or) R[UN]

enter value for sname: D

enter value for avrg: 78.92

INSERT INTO student VALUES(1004,'D',78.92)

1 row created.

/

Case-3: Inserting limited column values

SID	SNAME	AVRG
1005	E	

INSERT INTO student VALUES(1005,'E');

Output:

ERROR: not enough values

INSERT INTO student(sid, sname) VALUES(1005,'E');

1006	F	
-------------	----------	--

INSERT INTO student(sname, sid) VALUES('F',1006);

1007		44.55
-------------	--	--------------

INSERT INTO student(sid,avrg) VALUES(1007,44.55);

COMMIT;

SELECT * FROM student;

Example-2:

EMPLOYEE

EMPNO	ENAME	SAL	GENDER	DOJ
1001	AA	6000	M	25-DEC-22

empno	number(4)
ename	varchar2(10)
sal	max sal: 100000.00 number(8,2)
gender	CHAR(1)
doj	date

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

1001	AA	6000	M	25-DEC-22
------	----	------	---	-----------

default oracle date format: **DD-MON-YY**

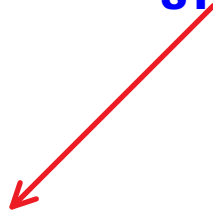
INSERT INTO employee
VALUES(1001,'AA',6000,'M','25-DEC-2022');
STRING

DOJ

25-DEC-22

DATE

Implicit conversion



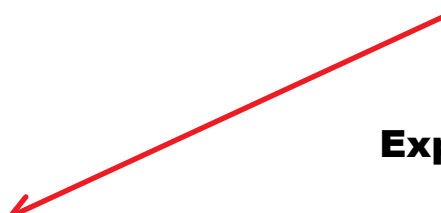
1002	BB	7000	F	17-AUG-21
------	----	------	---	-----------

INSERT INTO employee
VALUES(1002,'BB',7000,'F',To_Date('17-AUG-2021'));
STRING

DOJ

17-AUG-21 **DATE**

Explicit Conversion



Note:

ORACLE supports to 2 types of conversions. They are:

- **implicit conversion**
- **explicit conversion**

implicit conversion:

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

Example:

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

string string

↓ ↓

100 + 200 => 300

num num

implicit conversion

explicit conversion:

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

Example:

```
SELECT TO_NUMBER('100')+TO_NUMBER('200')
```

```
FROM dual;
```

Output:

300

Explicit
conversion

Don't depend on implicit conversion for 2 reasons:

- degrades the performance
- in further versions they may remove implicit conversion programs or they may change code of those programs

1003	CC			
------	----	--	--	--

```
INSERT INTO employee(empno,ename)  
VALUES(1003,'CC');
```

1004	DD	10000	M	today's date
------	----	-------	---	--------------

```
INSERT INTO employee VALUES(1004,'DD',10000,'M',sysdate);
```

Example-3:

EMPLOYEE1

EMPNO	ENAME	LOGIN_DATE_TIME
1001	AA	6-MAR-24 10:30:0.0 AM

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

1001	AA	6-MAR-24 10:30:0.0 AM
------	----	-----------------------

```
INSERT INTO employee1
VALUES(1001,'AA','6-MAR-2024 10:30:0.0 AM');
```

string

implicit conversion

login_date_time

6-MAR-2024 10:30:0.0 AM timestamp

1002	B	6-MAR-2024 2:30:0.0 PM
------	---	------------------------

```
INSERT INTO employee1
VALUES(1002,'B',to_timestamp('6-MAR-2024 2:30:0.0 PM'));
```

STRING

Explicit conversion

login_date_time

6-MAR-2024 2:30:0.0 PM TIMESTAMP

1003	CC	today's date and time
-------------	-----------	------------------------------

INSERT INTO employee1
VALUES(1003,'CC',*sys*timestamp);

Column Alias:

- **It is used change column heading in display.**
- **"AS" keyword is used to give column alias.**
- **Using "AS" keyword is optional**

Syntax:

<column> [AS] <column_alias>

Examples:

```
ENAME AS A  
SAL*12 AS ANNUAL_SAL  
SAL*12 ANNUAL_SAL
```

- **column alias is temporary.**
- **It's scope is limited to that query only.**
- **To maintain case or to give alias name in multiple words specify column alias in double quotes.**

SELECT

Thursday, March 7, 2024 10:24 AM

DRL / DQL:

- **DRL => Data Retrieval Language**
- **DQL => Data Query Language**
- **retrieve => opening existing data [selecting data]**
- **Query => is a request that is sent DB Server**

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

Syntax:

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

**SQL
QUERIES
CLAUSES**

**ENGLISH
SENTENCES
WORDS**

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

Using SELECT command we can SELECT:

- **All Columns and All Rows**
- **Specific Columns and All Rows**
- **Select All Columns and Specific Rows**
- **Select Specific Columns and Specific Rows**

- **All Columns and All Rows:**

Display all columns and rows if emp table:

```
SELECT * FROM emp;
```

*	All Columns
---	--------------------

```
SELECT * FROM emp;
```

ORACLE rewrites above query as following:

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

Note:

*** will be replaced with all column names in table structure order.**

Specific Columns and All Rows:

Display all emp names and salaries:

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

All Columns and Specific Rows:

Display the emp records whose salary is 3000:

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

Specific Columns and Specific Rows:

Display enames and salaries of the emps whose salary is 3000:

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

SELECT *	All columns
SELECT ename,sal	Specific Columns
NO WHERE clause	All rows
WHERE sal=3000	Specific Rows

SELECT	<p>is used to specify column list</p> <p>Syntax: SELECT <column_list></p> <p>Example: SELECT ename,sal</p>
FROM	<p>is used to specify table list</p> <p>Syntax: FROM <table_list></p> <p>Examples: FROM emp FROM emp, dept</p>
WHERE	<ul style="list-style-type: none"> • is used to specify filter condition • it filters the rows • if condition is TRUE row will be selected. • WHERE clause condition will be applied on every row. <p>Syntax: WHERE <condition></p> <p>Example: WHERE sal=3000</p>

Display all emp names and salaries whose salary is 3000:

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

EMP

EMPNO	ENAME	JOB	SAL
1001	A	CLERK	3000
1002	B	MANAGER	5000
1003	C	SALESMAN	3000
1004	D	ANALYST	4000

Execution Order:

- FROM
- WHERE
- SELECT

FROM emp:

ORACLE selects entire emp table

EMP

EMPNO	ENAME	JOB	SAL
1001	A	CLERK	3000
1002	B	MANAGER	5000
1003	C	SALESMAN	3000
1004	D	ANALYST	4000

WHERE sal=3000:

This WHERE condition will be applied on every row

EMP

EMPNO	ENAME	JOB	SAL
1001	A	CLERK	3000
1002	B	MANAGER	5000
1003	C	SALESMAN	3000
1004	D	ANALYST	4000

WHERE sal=3000

3000=3000 T

5000=3000 F

3000=3000 T

4000=3000 F

EMPNO	ENAME	JOB	SAL
1001	A	CLERK	3000
1003	C	SALESMAN	3000

SELECT ename,sal:

- it selects the columns

ENAME	SAL
A	3000
C	3000

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 NOT IN BETWEEN AND NOT BETWEEN AND LIKE NOT LIKE IS NULL IS NOT NULL EXISTS ALL ANY
SET	UNION UNION ALL INTERSECT MINUS
Concatenation	

Arithmetic Operators:

In C/ Java:

2)5(2

Arithmetic Operators:

+	Addition
-	Subtraction
*	Multiplication
/	Divison

In C/ Java:

int/int = int

$5/2 = 2$

$5\%2 = 1$

$2)5(2$

4

1

In ORACLE SQL:

$5/2 = 2.5$

$\text{MOD}(5,2) = 1$

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
-------	------	-------

Calculate Annual Salary of all emps. Display output with following column headings:

ENAME	SAL	ANUUAL_SAL
-------	-----	------------

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

Output:

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

SMITH	800	9600
-------	-----	------

ALLEN	1600	19200
-------	------	-------

Calculate Annual Salary of all emps. Display output with following column headings:

ENAME SAL Annual Salary

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

Output:
ERROR

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

Output:

ENAME	SAL	Annual Salary
SMITH	800	9600
ALLEN	1600	19200

Calculate experience of all emps:

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

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

10% on sal => TA

20% on sal => HRA

5% on sal => TAX

GROSS = basic_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;

Relational Operators / Comparison Operators:

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

Syntax:

<column> <relational_operator> <value>

ORACLE SQL provides following Relational Operators:

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

Examples on Relational Operators:

Display the emp records whose salary is 3000:

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

Display all managers records:

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

MANAGER = manager FALSE

Output:

No rows selected

Note:

When all conditions are FALSE, no rows will be selected

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

MANAGER = MANAGER => TRUE

Note:

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

Display 7698 employee record:

```
SELECT *
FROM emp
WHERE empno=7698;
```

**Display the emp records who are working
in deptno 30:**

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

Display all emp records except managers:

```
SELECT *
FROM emp
WHERE job!='MANAGER';
```

CLERK	!=	MANAGER	T
MANAGER	!=	MANAGER	F

ANALYST != MANAGER T

Display all emp records except 30th dept emps:

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

**Display the emp records whose salary is
3000 or more:**

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

sal	WHERE sal>=3000
-----	-----
2500	2500>=3000 F
3500	3500>3000 T
3000	3000>=3000 T
5000	5000>3000 T
1000	1000>=3000 F

**Display the emp records whose sal is 1250
or less:**

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

Note:
Calendar order is ASCENDING ORDER

1-JAN-2023

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

Display the emp records who joined after 1981:

31-DEC-1981

1-JAN-1982

2-JAN-1982

.

.

hiredate > '31-DEC-1981'

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

Display the emp records who joined before 1981:

.

.

30-DEC-1980

31-DEC-1980

1-JAN-1981

< '1-JAN-1981'

SELECT ename,hiredate
FROM emp

WHERE hiredate<'1-JAN-1981';

Logical Operators:

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

ORACLE SQL provides 3 Logical operators. They are:

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

AND, OR operators are used to separate multiple conditions

AND	should satisfy all conditions
OR	at least 1 condition should be satisfied

c1 => condition1
c2 => condition2

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

Examples on Logical Operators:

Display all managers and clerks records:

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

Display the emp records who joined in 1981:

after 1981	hiredate>'31-DEC-1981'
before 1981	hiredate<'1-JAN-1981'

```
SELECT ename,hiredate
FROM emp
WHERE hiredate>='1-JAN-1981' AND hiredate<='31-DEC-1981';
```

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

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

Example:

Student

SID	SNAME	M1	M2	M3
1001	A	60	80	70
1002	B	50	30	60

Max marks: 100

Min marks: 40 each subject

Display passed students records:

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

Display failed students records:

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

**Display the emp records whose empnos are:
7499, 7698, 7900**

```
SELECT *  
FROM emp  
WHERE empno=7499 OR empno=7698 OR empno=7900;
```

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

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

Display the emp records whose salary between 2000 and 3000:

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

NOT:

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

Truth table:

CONDN	NOT(CONDN)
T	NOT(T) => F
F	NOT(F) => T

Examples on NOT:

Display all emp records except managers:

```
SELECT ename,job,sal
```

FROM emp
WHERE NOT(job='MANAGER');

(or)	job	NOT(job='MANAGER')

SELECT ename,job,sal	MANAGER	MANAGER=MANAGER => NOT(T) => F
FROM emp	CLERK	CLERK=MANAGER => NOT(F) => T
WHERE job!='MANAGER';	ANALYST	ANALYST=MANAGER => NOT(F) => T

Special Operators:

IN:

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

Syntax:

<column> IN(<value_list>)

Example:

sal IN(800,1250,3000)

If sal value is in LIST then condition is TRUE.

sal	sal IN(800,1250,3000)
-----	-----
1250	1250 is in list => TRUE
1000	1000 is not in list => FALSE

Display the emp records whose salary is 800 or 1250 or 3000:

SELECT ename,sal
FROM emp
WHERE sal IN(800,1250,3000);

(or)

SELECT ename,sal
FROM emp
WHERE sal=800 OR sal=1250 OR sal=3000;

Examples on IN:

Display all managers and clerks records:

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

(or)

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

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

**Display the emp records whose names are:
JAMES, MILLER, SCOTT**

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

Display the emp records who are working in deptno

10 and 30:

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

deptno	deptno IN(10,30)
-----	-----
20	20 F
10	10 T
40	40 F
30	30 T

Display all emp records except managers and clerks:

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

If job value is NOT IN list then condition is TRUE

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

BETWEEN AND:

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

Syntax:

<column> BETWEEN <lower> AND <upper>

Example:

sal BETWEEN 2000 AND 3000

Examples on Between AND:

Display the emp records whose salary is between 2000 and 3000:

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

(or)

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

sal	sal BETWEEN 2000 AND 3000
-----	-----
2500	2500 T
2000	2000 T
5000	5000 F
4000	4000 F
3000	3000 T

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 who joined in 1982,1983:

```
SELECT ename,hiredate
FROM emp
WHERE hiredate BETWEEN '1-JAN-1982' AND '31-DEC-1983';
```

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

What is the output of above query?

- A. displays whose sal between 2000 and 3000**
- B. No rows selected**
- C. ERROR**
- D. None**

Answer: B

Display the emp records whose salary is less than 2000 or more than 3000:

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

If sal is not b/w 2000 and 3000 then condition is TRUE

sal	sal NOT BETWEEN 2000 AND 3000
-----	-----
2500	2500 F
1000	1000 T
2800	2800 F
4000	4000 T

Display the emp records who are not joined in 1981
(or)

Display all emp records except the emps who joined in 1981:

```
SELECT ename,hiredate
FROM emp
WHERE hiredate NOT BETWEEN '1-JAN-1981' AND '21-DEC-1981';
```

LIKE:

- it is used to compare column value with text pattern
- to specify text pattern ORACLE SQL provides 2 wildcard chars. they are:

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

Syntax:

<column> LIKE <text_pattern>

Examples on LIKE:

Display the emp records whose names are started with S:

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

Display the emp records whose names are ended with S:

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

```
ename
-----
S
```

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 names are having 4 chars:

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

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

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

Display the emp records whose names are having 'A' letter:

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

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 who joined in DECEMBER month:

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

Display the emp records who are getting 3 digit 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 to compare column value with NULL.

Syntax:

<column> IS null

Examples:

Display the emp records who are getting commission as 500:

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

Display the emp records who are getting commission as NULL:
(or)

Display the emp records who are not getting commission:

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

Output:

no rows selected

NULL = NULL FALSE

Note:

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

```
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;
```

comm

```
500   IS NOT NULL => TRUE
null  IS NOT NULL => FALSE
```

Concatenation Operator:

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

Syntax:

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

Examples:

Display emp records as following:

SMITH earns 800

ALLEN earns 1600

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

Display all emp records as following:

SMITH works as CLERK

ALLEN works as SALESMAN

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

Display all emp records as following:

SMITH joined on 17-DEC-1980 and works as CLERK

**SELECT ename || ' joined on ' || hiredate || ' and works as ' || job
FROM emp;**

NULL

Wednesday, March 13, 2024 9:22 AM

NULL:

- **NULL means empty / blank.**
- **NULL is not equals to 0.**
- **NULL is not equals to space.**
- **When we don't know the value or when we are unable to insert the value we insert NULL.**
- **If NULL is participated in operation then result will be NULL.**

Select 100+200 FROM dual; --300

Select 100+200+null FROM dual; --NULL


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

Example:

STUDENT

SID	SNAME	M1 NUMBER(3)
1001	A	70

1002	B	80
1003	C	0
1004	D	




NULL

Unable to insert ABSENT

EMPLOYEE

EMPNO	ENAME	SAL
1001	A	10000
1002	B	15000
1003	C	



NULL

sal value is unknown

UPDATE

Wednesday, March 13, 2024 10:04 AM

UPDATE:

- It is DML command.
- It is used to modify table data.
- using UPDATE command we can:
 - modify single value of single record
 - modify multiple values of single record
 - modify specific group of records
 - modify all records
 - modify records using parameters

Syntax:

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

Examples on UPDATE:

- modify 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;
```

- modify multiple values of single record:

**modify job as manager, sal as 6000 to the employee
who se empno is 7369:**

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

- **modify specific group of records:**

Increase 20% on sal to all managers:

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

- **modify all records:**

Increase 1000 rupees salary to all emps:

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

Transfer all deptno 10 emps to 20:

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

**Increase 10% on sal to the emps whose annual salary is
more than 30000:**

```
UPDATE emp
```

```
SET sal=sal+sal*0.1  
WHERE sal*12>30000;
```

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

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

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

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

Set comm as null to the emps whose empnos are 7499, 7521:

```
UPDATE emp  
SET comm=null  
WHERE empno IN(7499,7521);
```

Note:

For null assignment we can use =

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

Example:

STUDENT

SID	SNAME	M1	M2	M3	TOTAL	AVRG
1001	A	50	80	70		
1002	B	40	30	90		

```
CREATE TABLE student
(
  sid NUMBER(4),
  sname VARCHAR2(10),
  m1 NUMBER(3),
  m2 NUMBER(3),
  m3 NUMBER(3),
  total NUMBER(3),
  avrg NUMBER(5,2)
);
```

1001	A	50	80	70		
1002	B	40	30	90		

```
insert into student(sid,sname,m1,m2,m3)
values(1001,'A',50,80,70);
```

--indirect way

```
insert into student
values(1002,'B',40,30,80,null,null);
```

--direct way

Note:

we can insert NULL value using 2 ways.

- 1st way: indirect way => insert limited column values
- 2nd way: direct way => use NULL keyword

calculate total and avrg:

UPDATE student
SET total=m1+m2+m3;

UPDATE student
SET avrg=total/3;

Assignment:

EMPLOYEE

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

calculate TA, HRA, TAX and GROSS and store the result in table.

10% on basic sal as TA
20% on basic sal as HRA
5% on basic sal as TAX
GROSS = bsal + TA + HRA - TAX

Updating records using parameters:

Example:

Increase 10% on sal to the empno 7499
20% 7698
15% 7900

Increase 10% on sal to the empno 7499

```
UPDATE emp  
SET sal=sal+sal* &per/100  
WHERE empno=&empno;
```

Output:
enter value for per: 10
enter value for empno: 7499

/

Output:
enter value for per: 20
enter value for empno: 7698

/

Output:
enter value for per: 15
enter value for empno: 7900

DELETE

Thursday, March 14, 2024 9:46 AM

DELETE:

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

Syntax:

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

Examples on DELETE:

delete an emp record whose empno is 7788:

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

delete all managers records:

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

delete all records:

DELETE FROM emp;

(or)

DELETE emp;

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

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

delete the emp records who joined in DECEMBER month:

**DELETE FROM emp
WHERE hiredate LIKE '%DEC%';**

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

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

ALTER

Thursday, March 14, 2024 10:02 AM

ALTER:

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

Note:

to change table data use **UPDATE**

to change table structure use **ALTER**

Using ALTER command we can:

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

Syntax:

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

Example:

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);

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));

DESC student

Output:

```
NAME          TYPE  
-----  
sid           NUMBER(4)  
sname         VARCHAR2(10)  
m1            NUMBER(3)  
m2            NUMBER(3)  
m3            NUMBER(3)
```

Renaming Column [rename m3 to maths]:

**ALTER TABLE student
RENAME COLUMN m3 TO maths;**

DESC student

Output:

```
NAME          TYPE  
-----  
sid           NUMBER(4)
```

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

Dropping a column [drop maths column]:

ALTER TABLE 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 [drop m1 and m2 columns]:

ALTER TABLE student DROP(m1,m2);

DESC student

Output:

NAME	TYPE

sid	NUMBER(4)
sname	VARCHAR2(10)

Modifying field size [increase ename field size from 10 to 20]:

ALTER TABLE student MODIFY ename VARCHAR2(20);

DESC student

Output:

NAME	TYPE

sid	NUMBER(4)
sname	VARCHAR2(20)

Can we decrease the field size?

Yes. we can decrease. But, we can decrease up to max string length in column

SNAME

RAJU	4
NARESH	6
SAI	3

we can decrease up to 6 only

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

ALTER TABLE student MODIFY sid CHAR(7);

DESC student

DESC student

Output:

NAME	TYPE

sid	CHAR(7)
sname	VARCHAR2(20)

DROP, FLASHBACK and PURGE

Friday, March 15, 2024 9:19 AM

DROP:

- It is DDL command.
- It is used to drop the table.
- When we drop the table, it goes to recyclebin.

Syntax:

DROP TABLE <table_name> [PURGE];

Example:

DROP TABLE employee;

Flashback:

- it is DDL command.
- It is used to restore the dropped table from recyclebin.

Syntax:

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

Example:

FLASHBACK TABLE employee TO BEFORE DROP;

PURGE:

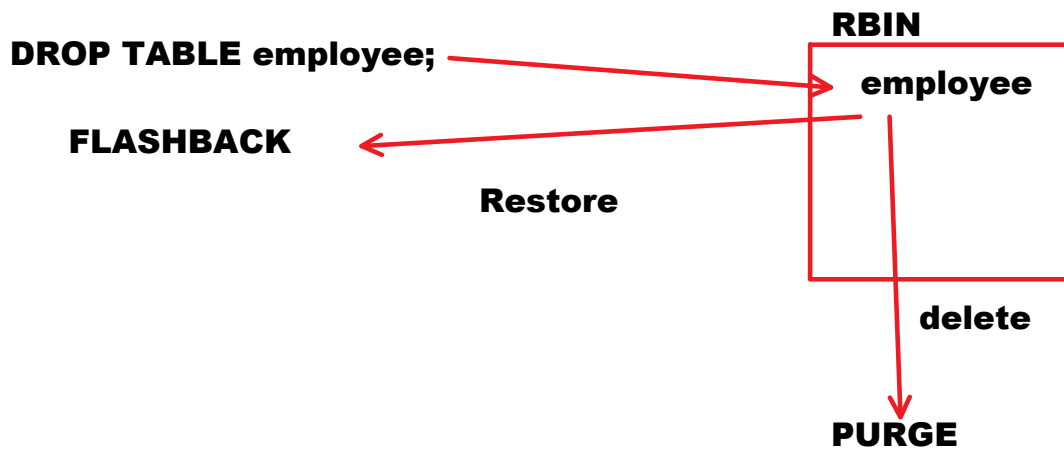
- It is DDL command.
- It is used to delete the table from recyclebin.
- If table is deleted from recyclebin it will be deleted permanently.

Syntax:

PURGE TABLE <table_name>;

Example:

PURGE TABLE employee;



Note:

- **RECYCLEBIN** feature added in **ORACLE 10g** version.
- **FLASHBACK** and **PURGE** commands introduced in **ORACLE 10g** version.

To see recyclebin:

SHOW RECYCLEBIN

Example on DROP, FLASHBACK and PURGE:

EMPLOYEE
5 columns
4 rows

deleting table:

DROP TABLE employee;

--table will be dropped. it will be placed in recyclebin

restoring table:

FLASHBACK TABLE employee TO BEFORE DROP;

deleting table from recyclebin (or) deleting table permanently:

DROP TABLE employee;

(or)

DROP TABLE employee PURGE;

PURGE TABLE employee;

--table will not be placed in recyclebin

--it will be deleted permanently

emptying recyclebin:

PURGE RECYCLEBIN;

Note:

don't practice recyclebin concept as "system" user.

Case-1:

CREATE TABLE t1(f1 int);

INSERT INTO t1 VALUES(1);

INSERT INTO t1 VALUES(2);

COMMIT;

DROP TABLE t1; --10:00 AM

RecycleBin

DROP TABLE t1; --10:00 AM

CREATE TABLE t1(f1 varchar2(10));

**INSERT INTO t1 VALUES('A');
INSERT INTO t1 VALUES('B');
COMMIT;**

DROP TABLE t1; --10:05 AM

RecycleBin

**t1 => 10.05 AM
t1 => 10.00 AM**

**FLASHBACK TABLE t1 TO BEFORE DROP;
--recent one will be restored [10:05 AM]**

to restore table dropped at 10:00 AM:

**FLASHBACK TABLE <recyclebin_table_name>
TO BEFORE DROP;**

Example:

**FLASHBACK TABLE "BIN\$kfMrGZptTqqOwLWh2Rruxw==\$0"
TO BEFORE DROP;**

Case-2:

CREATE TABLE t1(f1 int);

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

DROP TABLE t1;

CREATE TABLE t1(f1 VARCHAR2(10));

**INSERT INTO t1 VALUES('A');
INSERT INTO t1 VALUES('B');**

RBIN

T1 => 10.15

COMMIT;

**FLASHBACK TABLE t1
TO BEFORE DROP;**

Output:

**ERROR: name already used by existing
object**

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

Note:

SCHEMA => USER => c##batch9am

**Within SCHEMA table name must be
unique**

RENAME and TRUNCATE

- RENAME:**
- It is DDL command.
 - It is used to rename the table.

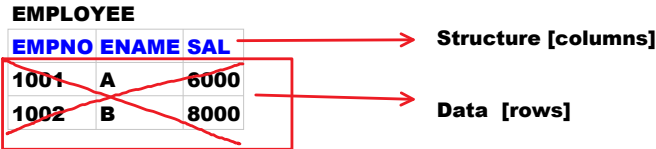
Syntax:

RENAME <old_name> **TO** <new_name>;

Example:

RENAME emp **TO** e;

TRUNCATE:



- It is DDL command.
- It is used to delete all rows with good performance.
- this data will be deleted permanently.
- it does not delete table structure. just, it deletes table data.

Syntax:

TRUNCATE TABLE <table_name>;

Example:

TRUNCATE TABLE employee;

--deletes all rows from employee table

What are the differences between TRUNCATE and DROP?

TRUNCATE	DROP
<ul style="list-style-type: none"> • it deletes all rows • it does not delete table structure • it cannot be flashed back 	<ul style="list-style-type: none"> • it deletes entire table • it deletes structure also. • it can be flashed back

TRUNCATE TABLE employee; --deletes all rows

DELETE FROM employee; --deletes all rows

What are the differences b/w TRUNCATE and DELETE?

TRUNCATE	DELETE
<ul style="list-style-type: none"> • Using it, we can delete all rows only. we cannot delete single row or specific group of rows. • WHERE clause cannot be used here • it is DDL command • it is auto committed • TRUNCATE = TRUNCATE+COMMIT It cannot be rolled back • It is faster • deletes block by block [page by page] 	<ul style="list-style-type: none"> • Using it, we can delete single record, specific group of records or all records. • WHERE clause can be used here. • it is DML command • it is not auto committed • it can be rolled back • it is slower • deletes row by row

Note:

All DDL commands are auto committed by default.

All DML commands are not auto committed by default.

DDL command = DDL command + COMMIT

Example:

CREATE = CREATE + COMMIT

ALTER = ALTER + COMMIT

TRUNCATE = TRUNCATE + COMMIT

CREATE TABLE t1(f1 int); => CREATE + COMMIT

INSERT INTO t1 VALUES(1);

INSERT INTO t1 VALUES(2);

CREATE TABLE t2(f1 VARCHAR2(10); => CREATE + COMMIT => committed

INSERT INTO t1 VALUES(3);

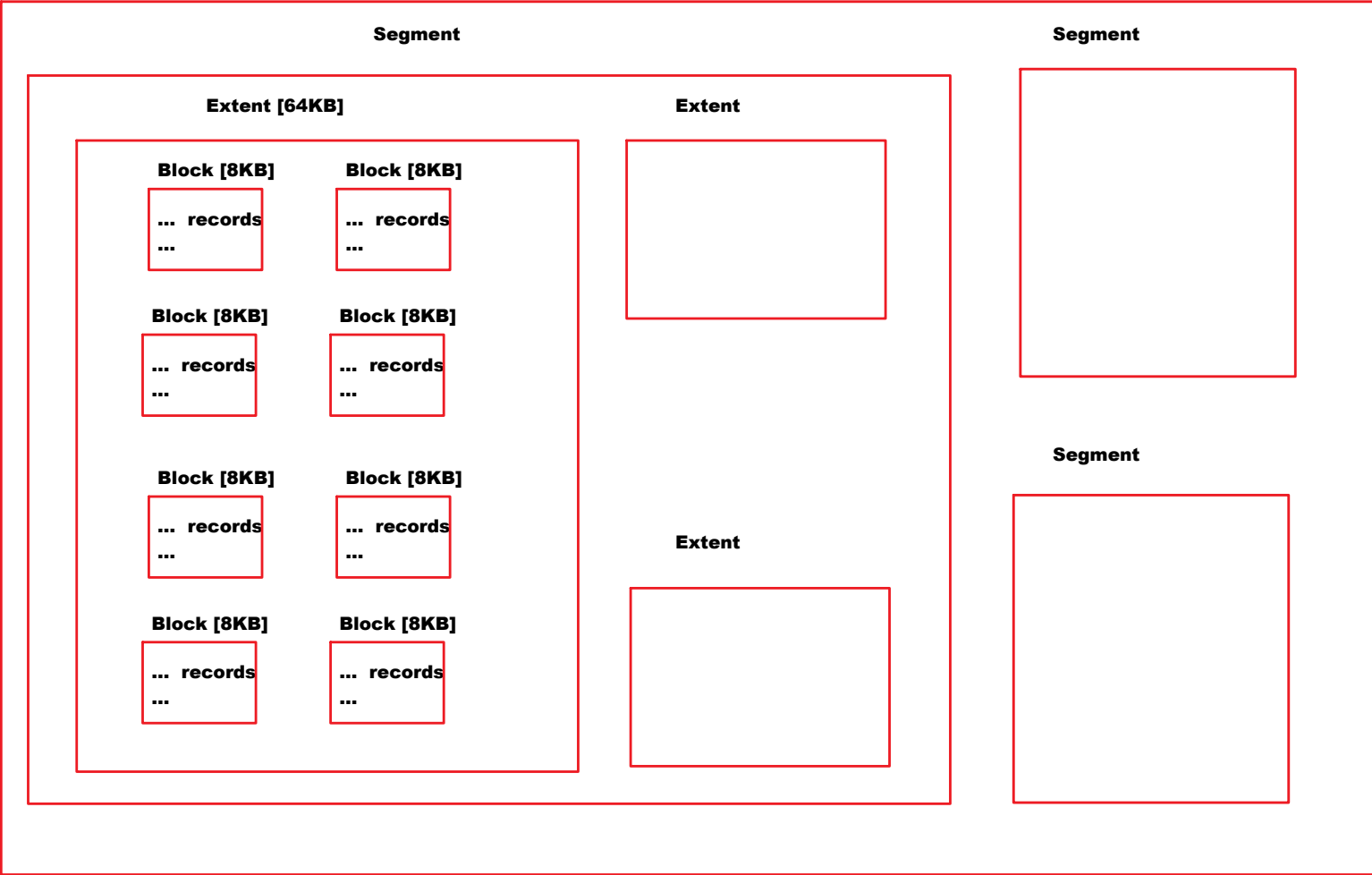
INSERT INTO t1 VALUES(4);

ROLLBACK; --cancels 2 uncommitted actions

Note:

Rollback cancels uncommitted actions

Tablespace



TableSpace
Segments
Extents
Blocks
Records

TCL

Friday, March 15, 2024 10:41 AM

TCL:

- **TCL => Transaction Control Language.**
- **Transaction => is a series of actions [SQL commands]**

Examples:

Withdraw, Deposit, Fund Transfer, Placing Order

ACCOUNTS

ACNO	NAME	BALANCE
1001	A	80000
1002	B	40000

Fund Transfer transaction:

BEGIN TRANSACTION

- **sufficient balance? => 20000**
- **update from account balance**
- **update to account balance**

END TRANSACTION

ACCOUNTS

ACNO	NAME	BALANCE
1001	A	60000
1002	B	60000

Note:

Transaction must be successfully finished or cancelled.

If transaction is successful, to save it use COMMIT

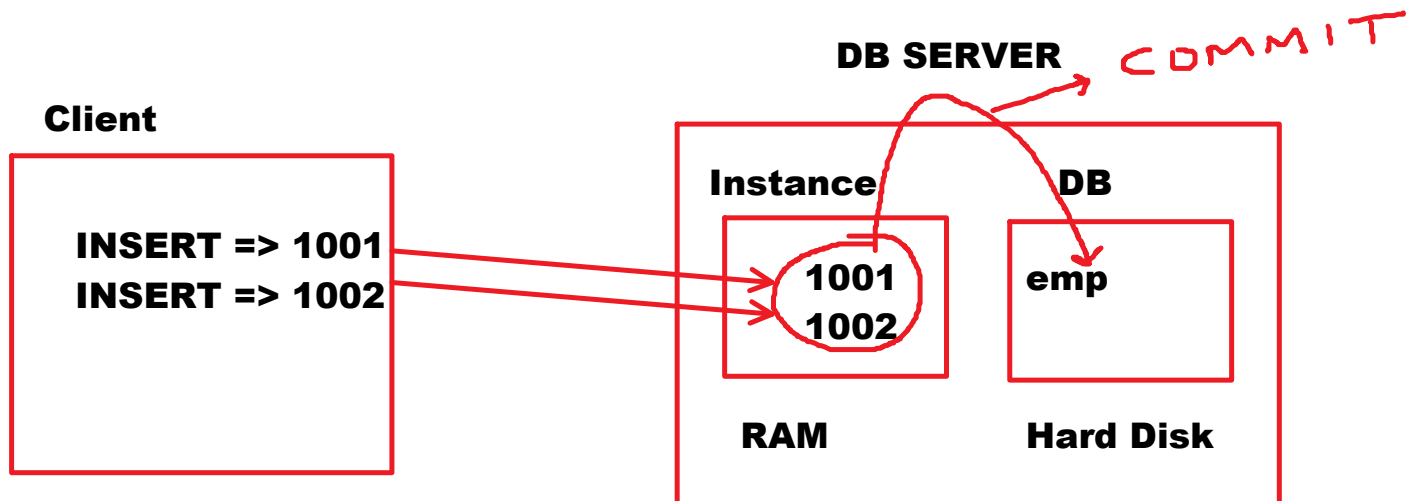
If transaction is unsuccessful, to cancel it use ROLLBACK

COMMIT [save]:

- **COMMIT command is used to save the transaction.**
- **When COMMIT command is executed, the changes in**

- INSTANCE [RAM] will be moved to DB [Hard Disk].**
- **COMMIT** command makes the changes permanent.

Syntax:
COMMIT;



ROLLBACK [undo]:

- **ROLLBACK** command is used to cancel the transaction.
- it cancels uncommitted actions.

Syntax:
ROLLBACK [TO savepoint_name>];

Example on COMMIT and ROLLBACK:

EMPLOYEE

EMPNO	ENAME	SAL
-------	-------	-----

100000.00

CREATE TABLE employee

**(
empno NUMBER(4),
ename VARCHAR2(10),
sal NUMBER(8,2)
);**

CREATE = CREATE + COMMIT

**INSERT INTO employee VALUES(1001,'A',6000); --inserts in INSTANCE
INSERT INTO employee VALUES(1002,'B',4000); --inserts in INSTANCE
COMMIT; -- 1001 and 1002 records will be moved to DB**

**INSERT INTO employee VALUES(1003,'C',9000); --inserts in INSTANCE
INSERT INTO employee VALUES(1004,'D',7000); --inserts in INSTANCE**

SELECT * FROM employee;

Output:

**1001 ..
1002 ..
1003 ..
1004 ..**

ROLLBACK; --cancel last 2 uncommitted actions

SELECT * FROM employee;

Output:

**1001 ..
1002 ..**

DELETE FROM employee WHERE empno=1001;

SELECT * FROM employee;

Output:

1002 ..

ROLLBACK;

SELECT * FROM employee;

Output:

1001 ..

1002 ..

Savepoint:

- **SAVEPOINT** is used to set margin for rollback.

Syntax:

Savepoint <savepoint_name>;

Example on Savepoint:

CREATE TABLE t1(f1 int);

--Begin Transaction => 10.00 AM

INSERT INTO t1 VALUES(1);
INSERT INTO t1 VALUES(2); => 10.10 AM

SAVEPOINT p1;

INSERT INTO t1 VALUES(3);
INSERT INTO t1 VALUES(4); => 10.20 AM

p2 — SAVEPOINT p2;

INSERT INTO t1 VALUES(5);
INSERT INTO t1 VALUES(6); => 10.30 AM

ROLLBACK TO p2; -- 2 actions will be cancelled

Note:

- **savepoint names are temporary.**
- **when transaction is finished, all savepoint names will be cleared.**

SQL

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

DCL

Monday, March 18, 2024 9:15 AM

DCL / ACL:

- **DCL => Data Control Language**
- **ACL => Accessing Control Language**
- **It deals with data accessibility.**
- **It is used to implement Table Level Security.**

Security can be implemented at 3 levels. They are:

- **Database Level => SCHEMA [USER]**
- **Table Level => DCL [Grant, Revoke]**
- **Data Level => VIEW**

ORACLE SQL provides 2 DCL commands. They are:

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

GRANT:

- **It is used to give permission on DB Objects [Tables, Views] to other users.**

Syntax:

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

privilege	permission
------------------	-------------------

Examples:

```
c##userA [owner]:  
emp
```

Grant read-only permission [select] to c##userB:

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

Grant write-only permission [DML permissions] to c##userB:

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

Giving all permissions to the user c##userB:

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

Giving read-only permission to c##userB, c##userC and c##userD:

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

Giving permission to all users [make table as public]:

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

REVOKE:

- It is used to cancel the permissions on Db objects from other users.

Syntax:

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

Examples:

```
c##userA [owner]:  
emp
```

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

cancel DML permissions:

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

cancel Read-only permission:

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

cancel all permissions:

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

Example on GRANT and REVOKE:

```
create 2 users c##userA, c##userB:
```

login as DBA:

username: system

password: naresh

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

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

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

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

Open 2 SQL PLUS windows.

Arrange them side by side [Windows + Right Arrow]

c##userA [GRANTOR]

c##userB [GRANTEE]

T1

F1	F2
1	A
2	B

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

**f2 VARCHAR2(10)
);**

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

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

SELECT * FROM c##userA.t1;

Output:

ERROR: table does not exist

SELECT * FROM c##userA.t1;

Output:

F1	F2
1	A
2	B

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;

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

**SELECT * FROM t1;
Output:**

F1	F2
1	A
2	B

**SELECT * FROM t1;
Output:**

F1	F2
1	A
2	B
3	C

**Output:
ERROR: Insufficient privileges**

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

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

F1	F2
1	A
2	B
3	C

COMMIT;

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

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

**REVOKE insert, update, delete
On t1
FROM c##userB;**

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

--all means 12 permissions

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

COMMIT;

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

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

F1	F2
2	B
3	C

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

ERRORL table does not exist

user_tab_privs_recd:

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

Example:

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

user_tab_privs_made:

- it is a system table / built-in table / readymade table
- it maintains all permissions list which are made by **GRANTOR**.

Example:

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

CASE:

c##userA	c##userB	c##batch9am
T1		
GRANT select ON t1 TO c##userB;		

TO c##userB;

**GRANT all
ON c##userA.t1
TO c##batch9am;
Output:
ERROR:**

**GRANT select
ON t1
TO c##userB
WITH GRANT OPTION;**

**Note:
WITH GRANT OPTION
clause is used to allow
GRANTEE to give
permission to other
users.**

**GRANT all
ON c##userA.t1
TO c##batch9am;
Output:
grant succeeded**

**--ALL means select
permission only. Because,
this user has only
SELECT permission**

**select *
from c##userA.t1;
--displays data**

Copying table and Copying records

Tuesday, March 19, 2024 9:16 AM

Copying table:

Syntax:

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

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

Example-1:

CREATE EXACT COPY OF EMP WITH THE NAME EMP1:

EMP

8 columns

16 rows

EMP1

8 columns

16 rows

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

Example-2:

Create a new table with the name emp2 from existing table emp with 4 columns and clerks records:

EMP

8 columns

16 rows

CLERK records

EMP2

empno, ename, job, sal

CLERKS records

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

Copying table Structure:

Syntax:

```
CREATE TABLE <table_name>
AS
SELECT <column_list> / *
FROM <table_name>
WHERE <false_condition>;
```

false_condition:

1=2

500=600

'A'='B'

```
FROM <table_name>  
WHERE <false_condition>;
```

500-600
'A'='B'

- To copy table structure we use **CREATE** command with **SELECT** query. But, in select query write false condition. Because, we don't want to copy any row.

Example-1:

EMP
8 columns
16 rows

EMP3
8 columns
no rows

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

Copying records:

Syntax:

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

- For copying records we use **INSERT** command with **SELECT** query.

- select query result will be inserted into table.

Example-1:



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

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

Example-2:



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



```
INSERT INTO emp5  
SELECT empno,ename,job,sal  
FROM emp  
WHERE job='MANAGER';
```

INSERT ALL

Tuesday, March 19, 2024 9:50 AM

INSERT ALL:

- It is DML command.
- Introduced in Oracle 9i version.
- It is used to copy one table records to multiple tables.
- It avoids of writing multiple insert commands.
- It is mainly used for ETL operations.

Extract - Transfer - Load

It can be used in 2 ways:

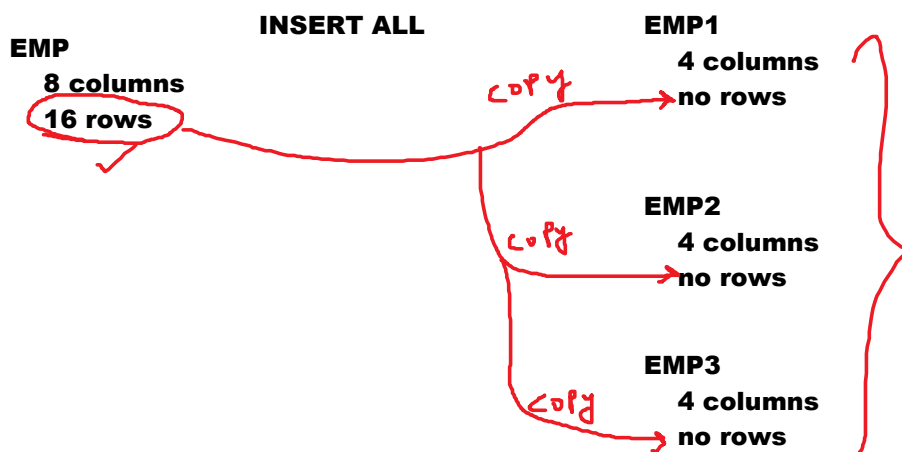
- Unconditional INSERT ALL
- Conditional INSERT ALL

Unconditional INSERT ALL:

Syntax:

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

Example:



Create emp1, emp2, emp3 tables with emp table structure and without rows:

```
CREATE TABLE emp1
AS
SELECT empno,ename,job,sal FROM emp
```

WHERE 1=2;

```
CREATE TABLE emp2  
AS  
SELECT empno,ename,job,sal FROM emp  
WHERE 1=2;
```

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

copy emp table all records to emp1, emp2 and emp3:

```
INSERT ALL  
INTO emp1 VALUES(empno,ename,job,sal)  
INTO emp2 VALUES(empno,ename,job,sal)  
INTO emp3 VALUES(empno,ename,job,sal)  
SELECT empno,ename,job,sal FROM emp;
```

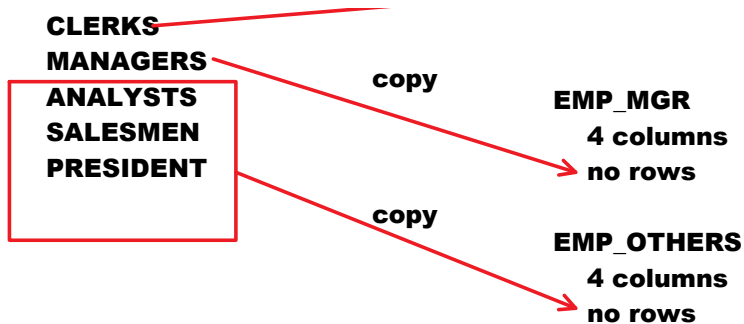
• **Conditional INSERT ALL:**

Syntax:

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

Example:

EMP		EMP_CLERK
8 columns		4 columns
16 rows	copy →	no rows
CLERKS		



Create emp_clerk, emp_mgr and emp_others with 4 columns without rows from existing table emp:

```

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

```

CREATE TABLE emp_mgr
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 clerks records to emp_clerk
managers records to emp_mgr
other than manager, clerk to emp_others:

```

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

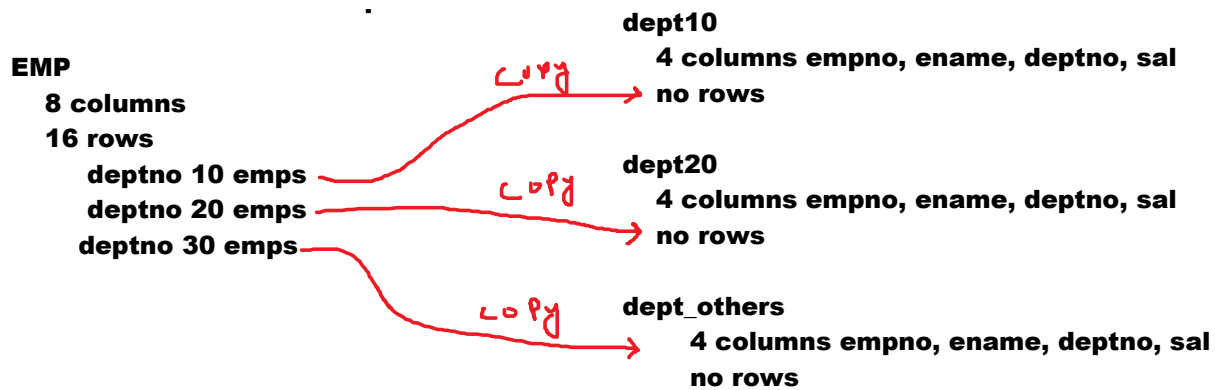
```

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

Assignment:

```

      WHEN deptno=10 THEN
                                dept10
EMP      4 columns empno, ename, deptno, sal
  8 columns
      copy → no rows
  
```



Note:

In INSERT ALL,
WHEN .. THEN is same as "if" control structure in C/Java.

In INSERT FIRST,
WHEN .. THEN is same as "if else if" control structure in C/Java.

MERGE

Wednesday, March 20, 2024 9:19 AM

MERGE:

Branch Office	S.CID = T.CID	Head Office																														
CUSTOMER1 S		CUSTOMER2 T [Replica => duplicate copy]																														
<table><tr><th>CID</th><th>CNAME</th><th>CCITY</th></tr><tr><td>1001</td><td>A</td><td>HYD BLR</td></tr><tr><td>1002</td><td>B</td><td>DLH HYD</td></tr><tr><td>1003</td><td>C</td><td>BLR</td></tr><tr><td>1004</td><td>D</td><td>CHENNAI</td></tr><tr><td>1005</td><td>E</td><td>PUNE</td></tr></table>	CID	CNAME	CCITY	1001	A	HYD BLR	1002	B	DLH HYD	1003	C	BLR	1004	D	CHENNAI	1005	E	PUNE		<table><tr><th>CID</th><th>CNAME</th><th>CCITY</th></tr><tr><td>1001</td><td>A</td><td>HYD</td></tr><tr><td>1002</td><td>B</td><td>DLH</td></tr><tr><td>1003</td><td>C</td><td>BLR</td></tr></table>	CID	CNAME	CCITY	1001	A	HYD	1002	B	DLH	1003	C	BLR
CID	CNAME	CCITY																														
1001	A	HYD BLR																														
1002	B	DLH HYD																														
1003	C	BLR																														
1004	D	CHENNAI																														
1005	E	PUNE																														
CID	CNAME	CCITY																														
1001	A	HYD																														
1002	B	DLH																														
1003	C	BLR																														

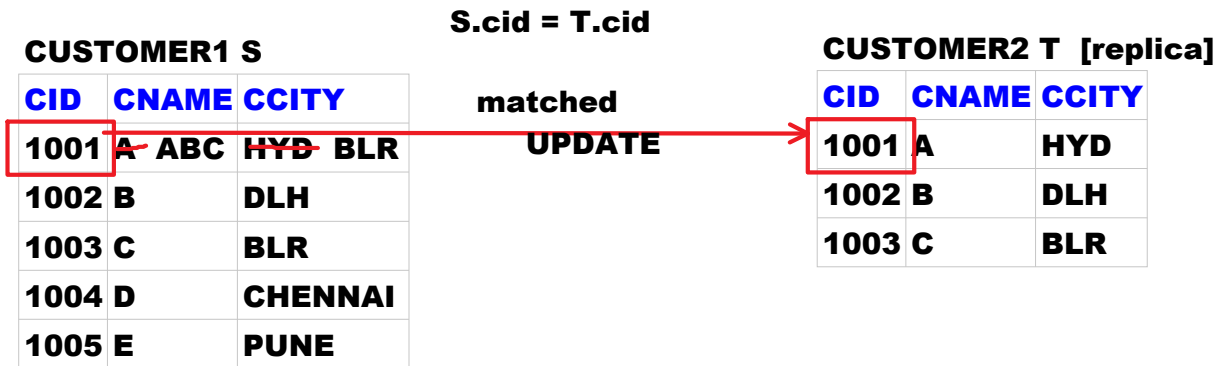
MERGE:

- Introduced in ORACLE 9i version.
- **MERGE** command is used to apply one table changes to it's replica [duplicate copy].
- It avoids of developing a separate PL/SQL program.
- **MERGE = UPDATE + INSERT**
- **MERGE** command is a combination of **UPDATE** and **INSERT**.
- It can be also called as "UPSERT" command.

Syntax:

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

Example:



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

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

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

1004	D	CHENNAI
1005	E	PUNE

```
INSERT INTO customer1 VALUES(1004,'D','CHENNAI');
INSERT INTO customer1 VALUES(1005,'E','PUNE');
COMMIT;
```

1001	A	ABC	HYD	BLR
------	---	-----	-----	-----

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

```
COMMIT;
```

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

Wednesday, March 20, 2024 10:42 AM

DUAL:

- it is a readymade table / system table.
- created in "SYS" schema [user].
- it has 1 column and 1 row.

DUAL

DUMMY

X

- Till ORACLE 21C version FROM clause is mandatory.
- to work with non-table data, to get 1 value as result we use table name DUAL.

Note:

- Till ORACLE 21C version FROM clause is mandatory.
- From ORACLE 23c version onwards writing FROM is optional.

In ORACLE 21C:

SELECT lower('KIRAN') FROM dual;

SELECT 100+200 FROM dual;

In ORACLE 23C:

SELECT lower('KIRAN');

SELECT 100+200;

Built-In Functions:

- **Built-In Functions =>**
predefined functions / system-defined functions / readymade functions
- **Function => Action / task / job**
- **To make our actions easier ORACLE DEVELOPERS already defined some functions and placed them in ORACLE DB. These functions are Built-In Functions.**

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

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

String Functions:

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

lower():

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

Syntax:

lower(<string>)

Examples:

lower('KIRAN')	kiran
lower('KIRAN KUMAR')	kiran kumar

SELECT lower('KIRAN') FROM dual;

output:

kiran

UPPER():

is used to convert the string to upper case.

Syntax:

UPPER(<string>)

Example:

UPPER('kiran')	KIRAN
-----------------------	--------------

Initcap(): [Initial Capital]

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

Syntax:

Initcap(<string>)

Examples:

Initcap('RAMU')	Ramu
Initcap('KIRAN KUMAR')	Kiran Kumar
Initcap('RAJ KUMAR VARMA')	Raj Kumar Varma

concat():

- **concatenate => combine**
- **used to combine 2 strings**

Syntax:

concat(<string1>, <string2>)

Examples:

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

length():

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

Syntax:**length(<string>)****Examples:**

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

Examples:

Display enames and salaries. display emp names in lower case:

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

Modify all emp names to lower case in emp table:

```
UPDATE emp  
Set ename=lower(ename);
```

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

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

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

Output:
no rows selected

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

Output: displays BLAKE record

```
lower(ename)='blake'
```

ename

SMITH	lower('SMITH')	smith = blake	FALSE
ALLEN	lower('ALLEN')	allen = blake	FALSE
BLAKE	lower('BLAKE')	blake = blake	TRUE

Display the emp records whose names are having 4 chars:

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

(or)

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

Display the emp records whose names are having 14 chars:

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

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

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

Example:

EMPLOYEE

EMPNO	FNAME	LNAME
1001	RAJ	KUMAR
1002	SAI	KRISHNA

ENAME

Raj Kumar

Sai Krishna

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');  
COMMIT;
```

Add ename column:

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

concatenate fname and last name by separating them using

space and place it in ename column in initcap case:

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

(or)

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

Drop fname and lname:

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

Substr():

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

Syntax:

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

Examples:

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

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

2nd argument [position] can be -ve

if 2nd is +ve	from left side
if 2nd arg is -ve	from right side

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

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

Example:

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

EMP

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

ALTER TABLE emp ADD mail_id varchar2(30);

UPDATE emp

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

Display the emp records whose name started with 'S':

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

(or)

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

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

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

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

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

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

```
SELECT *
```



```

FROM emp
WHERE Substr(ename,1,1) = Substr(ename,-1,1);

```

Display the emp records whose are started with VOWEL:

```

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

```

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

```

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

```

Instr():

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

Syntax:

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

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

Examples:

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

3rd arg position number can be -ve

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

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

Display the emp records whose names are having AM chars:

```

SELECT *
FROM emp

```

WHERE Instr(ename,'AM')>0;

(or)

SELECT *
FROM emp
WHERE ename LIKE '%AM%';

ename

AMAR
JAMES
ADAMS
WARD
SMITH

Display the emp names which are having _ :

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

(or)

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

(or)

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

Example:

EMPLOYEE

EMPNO	ENAME
1001	RAJ KUMAR
1002	RAVI TEJA
1003	SAI KIRAN

FNAME	LNAME
RAJ	KUMAR
RAVI	TEJA
SAI	KRISHNA

```
create table employee
(
  empno number(4),
  ename varchar2(10)
);
```

```
insert into employee values(1001,'RAJ KUMAR');
insert into employee values(1002,'RAVI TEJA');
insert into employee values(1003,'SAI KIRAN');
COMMIT;
```

adding fname and lname columns:

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

extract first name from ename and place it in fname:

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

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

drop ename column:

```
ALTER TABLE employee DROP COLUMN ename;
```

Lpad() and Rpad():

- pad => fill

Lpad():

- is used to fill specific char set at left side.

Syntax:

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

3rd arg	char	default value	space
---------	------	---------------	-------

Rpad():

is used to fill specific char set at right side.

Syntax:

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

char set	char/chars
----------	------------

3rd arg	char	default value	space
---------	------	---------------	-------

Examples:

Lpad('RAJU',10,'*')	10-4 = 6	*****RAJU
Rpad('RAJU',10,'*')	10-4 = 6	RAJU*****
Lpad('SAI',8,'#')	8-3 = 5	#####SAI
Rpad('SAI',8,'#')	8-3 = 5	SAI#####
Lpad('SAI',10,'\$@')	10-3 = 7	\$@\$@\$@\$SAI
Lpad('RAVI',10)	10-4 = 6	6spacesRAVI
Rpad('RAVI',10)	10-4 = 6	RAVI6spaces

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

Example:

Accounts

ACNO	BALANCE
1234567891	50000

display output as following:

amount debited from XXXXXX7891

```
SELECT 'amount debited from ' || Lpad('X',6,'X') ||
Substr('1234567891',-4,4) FROM dual;
```

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

- Trim => Remove
- To remove unwanted chars we use TRIM functions

Ltrim():

- used to remove unwanted chars from left side.

Syntax:

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

2nd arg	char	default value	space
---------	------	---------------	-------

Rtrim():

- used to remove unwanted chars from right side.

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

2nd arg	char	default value	space
---------	------	---------------	-------

Trim():**used to unwanted chars from left side or right side or both sides.****Syntax:****Trim(Leading / Trailing / Both <char> FROM <string>)****default side is BOTH****default char is space****Examples:**

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

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

Replace():

- is used to replace search string with replace string.

Syntax:**Replace(<string>, <search_string>, <replace_string>)****Examples:**

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

Translate():

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

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

Examples:

Replace('SAI KRISHNA','SAI','XYZ')	XYZ KRISHNA
Translate('SAI KRISHNA','SAI','XYZ')	XYZ KRZXHNY
S => X	
A => Y	
I => Z	

Replace('abcaabbccabc','abc','XYZ')	XYZaabbccXYZ
Translate('abcaabbccabc','abc','XYZ')	XYZXXYYZZXYZ
a => X	
b => Y	
c => Z	

**What is the difference between Replace()
& Translate():**

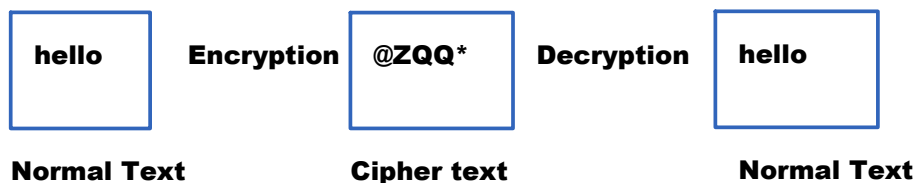
Replace()	replaces the strings
Translate()	replaces the chars

Note:

Translate can be used to encrypt the data.

Encryption: Normal Text => Cipher Text

Decryption: 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	Q	W	^	@	*	B	Y	X

**SELECT ename, Translate(sal,'0123456789','#ZQW^@*BYX') AS sal
FROM emp;**

```
Replace('RA***JU','***','') RAJU
```

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

string string

100 + 200
number number

Implicit conversion

Output:

300

Note:

don't depend on Implicit Conversion for 2 reasons:

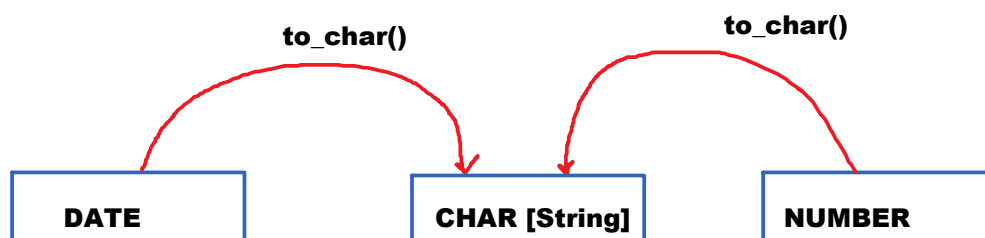
- it degrades the performance
- in further versions they may remove implicit conversion programs

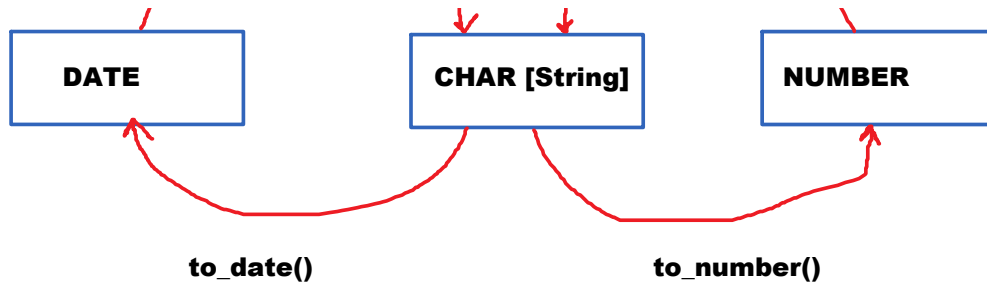
Explicit Conversion:

- if conversion is done by Built-In Function then it is called "Explicit Conversion".

ORACLE SQL provides following Built-In Functions:

- To_Char()
- To_Date()
- To_Number()





To_Char() [Date to Char]:

- can be used to convert date to char.

- to change date formats we can use it.

default ORACLE date format: DD-MON-YY

to_Char()
DD/MM/YYYY
IND date format

- to get part of the date like only date, only year, only month ..etc

Syntax:

To_Char(<date>, <format>)

FORMAT	PURPOSE	Example sysdate => 25-MAR-24	Output
YYYY	year 4 digits	to_char(sysdate,'YYYY')	2024
YY	year last 2 digits	to_char(sysdate,'YY')	24
YEAR / year	year in words	to_char(sysdate,'YEAR') to_char(sysdate,'year')	TWENTY TWENTY-FOUR twenty twenty-four
MM	month number	to_char(sysdate,'MM')	03
MON / mon	short month name	to_char(sysdate,'MON') to_char(sysdate,'mon')	MAR mar
MONTH	full month name	to_char(sysdate,'MONTH "')	MARCH
D	day num in week 1=> sun 2=> mon . . 7=> sat	to_char(sysdate,'D')	2

DD	day num in month	to_char(sysdate,'DD')	25
DDD	day num in year	to_char(sysdate,'DDD')	85 31+29+25 = 85
DY	short weekday name SUN MON . SAT	to_char(sysdate,'DY')	MON
DAY	full weekday name	to_char(sysdate,'DAY')	MONDAY
Q	quarter number 1 => jan-mar 2 => apr-jun 3 => jul-sep 4 => oct-dec	to_char(sysdate,'Q')	1
CC	century number	to_char(sysdate,'CC')	21
AD / BC	AD or BC	to_char(sysdate,'BC')	AD
HH / HH12	hours in 12 hrs format		
HH24	hours in 24 hrs format		
MI	minutes part		
SS	seconds part		
FF	fractional seconds		
AM / PM	AM or PM		

Display current system date:

SELECT sysdate FROM dual;

Display current system date and time:

SELECT systimestamp FROM dual;

Display current system time in 12hrs format:

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

Display current system time in 24hrs format:

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

Display current system time in 12hrs format including fractional seconds:

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

Display the emp records who joined in 1982:

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

(or)

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

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

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

Display the emp records who joined in 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';
```

(or)

```
SELECT ename,hiredate  
FROM emp  
WHERE to_char(hiredate,'MONTH')='DECEMBER';  
DECEMBERspace = DECEMBER FALSE  
Output: no rows selected
```

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

(or)

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

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

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

Display the emp records who joined on 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';
      SUNDAY3spaces = SUNDAY   FALSE
```

Output:

no rows selected

```
select ename,hiredate
from emp
where rtrim(to_char(hiredate,'DAY'))='SUNDAY';
```

(or)

```
select ename,hiredate
from emp
where to_char(hiredate,'DAY')='SUNDAY  ';
```

NOTE:

for every weekday to_char() function returns 9 chars

when we use DAY format.

SUNDAY	SUNDAY3spaces
MONDAY	MONDAY3space
TUESDAY	TUESDAY2spaces
WEDNESDAY	WEDNESDAY
..	
..	
SATURDAY	SATURDAY1space

Display the emp records who joined in first quarter [jan, feb, mar]:

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

Display the emp records who joined in first and fourth quarter:

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

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

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

To_Char() [number to char]:

- Using to_char() we can convert number to char [string].
- To apply currency symbols, currency names, decimal point, decimal places and thousand separator we can use it.

Syntax:

To_Char(<number> [, <format> , <nls_parameters>])

Examples:

To_Char(123)	'123'
To_Char(123.45)	'123.45'

FORMAT	PURPOSE
L	Currency Symbol Example: \$
C	Currency Name Example: USD
. / D	Decimal point
9	Digit
, / G	Thousand Separator

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

Examples:

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

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

sal	sal
-----	-----
1600	\$1600.00

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

```
SELECT ename, To_Char(sal,'C99,999.99') AS sal  
FROM emp;
```

sal	sal
-----	-----
1600	USD1,600.00

Note:

To see NLS parameters list:

Login as DBA:

username: system

password: naresh

SQL> show parameters --displays all parameters

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

NLS_CURRENCY	\$
NLS_ISO_CURRENCY	AMERICA

Convert 5000 number as ¥5000.00:

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

FROM dual;

Output:

¥5000.00

Convert 5000 number as JPY5000.00:

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

FROM dual;

Output:

JPY5000.00

Display all emp names and salaries.

Apply ¥ currency symbol to salaries:

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

FROM emp;

Display all emp names and salaries.

Apply RS currency symbol to salaries:

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

FROM emp;

Display all emp names and salaries.

Apply Currency name INR:

SELECT ename, to_char(sal,'C99999.99','NLS_ISO_CURRENCY=INDIA') AS sal

FROM emp;

To_Date():

- is used to convert string to date.
 - it can be used to insert DATE values.
 - To get part of the date from specific date.

Syntax:

To_Date(<date> [, <format>])

Examples:

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

Example:

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

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

STRING

F1 DATE

25-DEC-23 DATE

Implicit Conversion

Note:

implicit conversion degrades performance

```
INSERT INTO t1 VALUES(To_Date('17-AUG-2020'));
```

STRING

F1

17-AUG-20 DATE

Explicit Conversion

```
INSERT INTO t1 VALUES(To_Date('25/10/2023','DD/MM/YYYY'));
```

```
INSERT INTO t1 VALUES(To_Date('&d/&m/&y','DD/MM/YYYY'));
```

Output:

enter ... d:18

enter ... m:11

enter ... y:2022

/

Output:

enter ... d:..

enter ... m:..

enter ... y:..

Find today's weekday name:

SELECT to_char(sysdate,'DAY') FROM dual;
date

D	1 to 7
DY	SUN
DAY	SUNDAY

Find the weekday on which INDIA Independence:

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

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

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.

Syntax:

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

Examples:

to_number('123')	123
to_number('123.45')	123.45
to_number('\$5000.00')	ERROR
to_number('\$5000.00','L9999.99')	5000
to_number('USD5,000.00','C9,999.99')	5000

Aggregate Functions / Group Functions / Multi Row Functions:

40
90
20
30

sum	40+90+20+30 = 180
avg	180/4 = 45
max	90
min	20
count	4

sum():

it 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 and 30:

**SELECT sum(sal) FROM emp
WHERE deptno IN(10,30);**

avg():

is used to find average of a set of values.

Syntax:

avg(<column>)

Examples:

Find avrg salary of all emps:

SELECT avg(sal) FROM emp;

Find avrg salary of all CLERKS:

SELECT avg(Sal) FROM emp

WHERE job='CLERK';

max():

is used to find maximum 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 managers:

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

min():

is use to find minimum value in a set of values

Syntax:

min(<column>)

Examples:

Find min salary in all emps:

SELECT min(sal) FROM emp;

Find min salary in deptno 30:

**SELECT min(sal) FROM emp
WHERE deptno=30;**

count():

- **is used to count no of records or number of column values.**

Syntax:

count(* / <column>)

Examples:

Find no of records in emp table:

SELECT count(*) FROM emp;

Find no of managers:

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

Find no of emps in deptno 30:

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

Find how many emps are getting comm:

SELECT count(comm) FROM emp;

Difference b/w count(*) and count(<any number>):

count(*)	counts the records slower
count(8)	counts the 8s faster

Date Functions:

**sysdate
systimestamp**

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

**sysdate:
it is used to get current system date**

**Example:
display today's date:**

select sysdate from dual;

**systimestamp:
it is use to get current system date and time**

**Example:
display today's date and present time:**

select systimestamp from dual;

Add_Months():

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

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

Subtract 2 days from today's date:

SELECT sysdate-2 FROM dual;

Subtract 2 months from today's date:

SELECT add_months(sysdate,-2) FROM dual;

Subtract 2 years from today's date:

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

ORDERS

ORDER_ID	CID	ordered-date	delivery_date
123456	1234	sysdate	sysdate+5

PRODUCTS

PID	PNAME	MANUFACTURED_DATE	EXPIRY_DATE
1001	XYZ	sysdate	Add_Months(sysdate,3)

CM_LIST

STATE_CODE	CM_NAME	START_DATE	END_DATE
TS	RR	9-DEC-23	Add_months(start_date,5*12)

1

DOB => Date Of Birth
DOR => Date Of Retirement

EMPLOYEE

EMPID	ENAME	DOB	DOR
1001	A	25-DEC-2000	Add_Months(DOB,60*12)

Example:

```
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;
27-MAR-24 10:36 = 27-MAR-24 10:43 FALSE
```

Output:
No rows selected.

```
SELECT ename,hiredate
FROM emp
WHERE TRUNC(hiredate)=TRUNC(sysdate);
27-MAR-24 = 27-MAR-24 TRUE
```

```
select ename,hiredate
from emp
where to_char(hiredate,'DD/MM/YYYY')=to_char(sysdate,'DD/MM/YYYY');
```

Note:
To remove time from date and time we use TRUNC function

SELECT systimestamp FROM dual;

Output:

27-MAR-24 10.45.05.123456 AM +5.30

SELECT TRUNC(systimestamp) FROM dual;

Output:

27-MAR-24

Display the emp records who joined yesterday:

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

Display the emp record who joined 1 month ago from today's date:

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

Display the emp record who joined 1 year ago from today's date:

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

Assignment:

SALES

DATEID	AMOUNT
1-JAN-2023	50000
2-JAN-2023	70000
..	
27-MAR-24	60000

find today's sales

find yesterday's sales

find 1 month ago sales

find 1 year ago sales

GOLDRATE

DATEID	PRICE
1-JAN-23	50000
2-JAN-23	50500
..	
..	
27-MAR-24	61000

Find today's gold rate

Find yesterday's gold rate

Find 1 month ago gold rate

Find 1 year ago gold rate

Last_Day():

- it is used to get last date in the month.

Syntax:

Last_day(<date>)

Examples:

Last_Day(sysdate)	31-MAR-24
Last_Day('17-FEB-2024')	29-FEB-24
Last_Day('17-FEB-2023')	28-FEB-24

Find next month first date:

```
SELECT Last_day(sysdate)+1 FROM dual;
          31-MAR-24 + 1
          1-APR-24
```

Find current month first date:

```
SELECT Last_Day(Add_Months(sysdate,-1))+1 FROM dual;
          28-FEB-24
          29-FEB-24+1
          1-MAR-24
```

Next_Day():

- it is used to get next date based on the weekday.

Syntax:

Next_Day(<date>, <weekday_format>)

D	1
DY	SUN
DAY	SUNDAY

Examples:**Find next Sunday date:**

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

sysdate => 28-MAR-24

31-MAR-24

Find next WDNESDAY DATE:

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

Find next month first Sunday date:

SELECT Next_day>Last_day(sysdate),'SUN') FROM dual;
31-MAR-24

Find current month last Sunday date:

SELECT Next_day>Last_Day(sysdate)-7,'SUN') FROM dual;

Months_Between():

- it is used to get difference between 2 date values.

Syntax:

Months_Between(<date1>, <date2>)

Example:

Months_Between('1-Jan-2024','1-JAN-2023')	12
Months_Between('1-Jan-2024','1-JAN-2023')	12/12 = 1

Find experience of all employees:

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

(or)

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

Display experience of all emps in the form of years and months:

emp has 15 months

YEAR MONTHS

1

3

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

Number Functions:

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

power():

- is used to find power value.

Syntax:

power(<number>, <power>)

Example:

power(2,3)	8
-------------------	----------

sqrt():

is used to find square root value

Syntax:

sqrt(<number>)

Example:

sqrt(100)	10
------------------	-----------

sign():

- is used to check whether a number is +ve or -ve or zero.
- 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():

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

Syntax:

abs(<number>)

Examples:

abs(25)	25
abs(-25)	25

ceil():

is used to get round up value

Syntax:

ceil(<number>)

floor():

is used to get round down values

Syntax:

floor(<number>)

Examples:

ceil(567.89234)	568
floor(567.89234)	567

567 => 567.89234 => 568

Find avg salary of all emps. take round up value:

SELECT Ceil(avg(Sal)) FROM emp;

Find avg salary of all emps. take round down value:

SELECT Floor(avg(Sal)) FROM emp;

TRUNC():

- is used to remove the decimal places.

Syntax:

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

Examples:

TRUNC(123.6789)	123
TRUNC(123.6789,1)	123.6
TRUNC(123.678924,3)	123.678

Note:

- 2nd argument can be -ve also.
- If 2nd argument is -ve, it will not give decimal places

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

Examples:

TRUNC(786.6789,-1)	780 and 790 780
TRUNC(8786.6789,-1)	8780 and 8790 8780
TRUNC(8786.6789,-2)	8700 and 8800 8700
TRUNC(8786.6789,-3)	8000 and 9000 8000
TRUNC(786.6789,-2)	700 and 800 700

Round():

- is used to get rounded values.
- if value is avrg or above avrg, it gives upper value.
if value is below avrg, it gives lower value.

Syntax:

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

Examples:

Round(456.678)	456 => 456.678 => 457 avrg: 456.5 457
Trunc(456.678)	456
Round(456.345)	456 => 456.678 => 457 avrg: 456.5 456
Trunc(456.345)	456
Round(456.5)	456 => 456.678 => 457 avrg: 456.5 457
TRUNC(456.5)	456

Note:

Trunc() does not consider avrg.
it always gives lower value

Round() considers avrg.
if value is avrg or above avrg, it gives upper value.
if value is below avrg, it gives lower value.

ROUND(456.34789,2)	456.35
TRUNC(456.34789,2)	456.34
ROUND(456.34389,2)	456.34
TRUNC(456.34389,2)	456.34

Round(167.4567,-1)	160 => 167.4567 => 170 avrg: 165 170
Trunc(167.4567,-1)	160 => 167.4567 => 170 160
Round(163.4567,-1)	160 => 163.4567=> 170 avrg: 165 160
Trunc(163.4567,-1)	160 => 163.4567=> 170 160
Round(786.678,-2)	700 => 786.678 => 800 avrg: 750

	800
TRUNC(786.678,-2)	700 => 786.678 => 800 700
Round(4678.3456,-3)	4000 => 4678.3456 => 5000 avrg: 4500 5000
TRUNC(4678.3456,-3)	4000 => 4678.3456 => 5000 4000

Mod():
it is used to get remainder value

Syntax:
Mod(<number>, <divisor>)

Examples:

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

Analytic Functions / Window Functions:

Rank()
Dense_Rank()
Row_Number()

Rank():

- used to apply ranks to records according to specific column order.
- It does not follow sequence in ranking if multiple values are same.

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

Dense_Rank():

- used to apply ranks to records according to specific column order.
- It always follows sequence in ranking even if multiple values are same.

Syntax:

**Dense_Rank() OVER(PARTITION BY <column>
ORDER BY <Column> ASC/DESC)**

Example:**STUDENT****ORDER BY marks DESC**

MARKS
500
930
750
840
930
750
840
750
600
400

MARKS	RANK	DENSE RANK
930	1	1
930	1	1
840	3	2
840	3	2
750	5	3
750	5	3
750	5	3
600	8	4
500	9	5
400	10	6

dense	no gaps
--------------	----------------

Examples on Rank(), Dense_Rank():

Apply ranks to all emp records. 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 all emp records. give top rank to most senior:

ORDER BY hiredate ASC

hiredate

25-DEC-2020
17-AUG-2018

hiredate

17-AUG-2018
25 DEC 2020

min date => max experience

-----	-----	
25-DEC-2020	17-AUG-2018	min date => max experience
17-AUG-2018	25-DEC-2020	
10-FEB-2023	10-FEB-2023	

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

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

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

Example: dense_rank() over(PARTITION BY deptno
ORDER BY sal DESC)

EMP

EMPNO	ENAME	DEPTNO	SAL
1001	A	10	6000
1002	B	10	10000
1003	C	20	20000
1004	D	20	15000

10	10000	1
10	6000	2
20	20000	1
20	15000	2

Apply ranks to emp records.
within dept give top rank to highest salary:
[dept wise, with in dept salary wise desc apply ranks]

break on deptno skip 1

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

Apply ranks to emp records.
within dept give top rank to senior:
[dept wise, within dept seniority wise apply ranks]

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

Apply ranks to records.

Within Job, according to salary descending order apply ranks:

```
SELECT ename, job, sal,  
dense_rank() OVER(PARTITION BY job ORDER BY sal DESC) as rank  
FROM emp;
```

Row_Number():

- 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 according to empno ascending order:

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

Miscellaneous Functions:

NVL()

NVL2()

GREATEST()

LEAST()

USER

UID

NVL():

- is used to replace 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 100+200:**SELECT 100+200 FROM dual;****Output:****300****calculate 100+200+null:****SELECT 100+200+NULL FROM dual;****Output:****NULL => wrong output****SELECT 100+200+NVL(null,0) FROM dual;****Output:****300****Examples on NVL():****Find total salary of emps [sal+comm]:****SELECT ename, sal, comm, sal+NVL(comm,0) AS "total salary"
FROM emp;****Display all emp records along with commissions.****If comm is null display it as N/A [Not Applicable]:****SELECT ename, sal, NVL(comm,'N/A') AS comm
FROM emp;****Output:****ERROR: data types are mismatching****comm => number type****N/A => char type**

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

Example:

STUDENT

SID	SNAME	M1
1001	A	70
1002	B	0
1003	C	
1004	D	50
1005	E	
1006	F	44

display all emp records.
if m1 value is null display it as ABSENT

```
select sid,sname,
nvl(to_char(m1),'ABSENT') AS m1
FROM student;
```

NVL2():

- is used to replace nulls and not nulls.

Syntax:

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

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

Examples:

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

Example:

increase 1000 rupees comm to the emps who are getting commission. if emp is not getting commission set comm as 900:

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

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

NVL()	<ul style="list-style-type: none"> • can replace nulls only • can take 2 arguments
-------	--

NVL2()	can replace nulls and not nulls • can take 3 arguments
---------------	---

Max()	• used to find max value in vertical values [column] • Syntax: max(<column>) • it can take 1 argument • it is multi row function
Greatest()	• used to find max value in horizontal values [row] • Syntax: greatest(<arg1>,<arg2>, ...,<arg_n>) • can take variable length arguments • it is single row function

sal

7000
5000
8000

max(sal)

F1	F2	F3
70	90	50
80	55	77

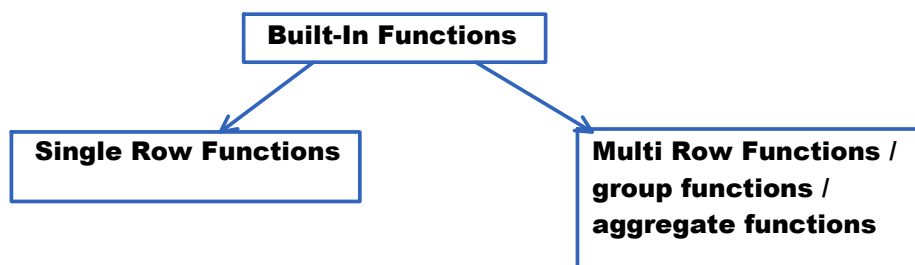
Greatest(f1,f2,f3)

Greatest(70,90,50) => 90

Greatest(80,55,77) => 80

max() one function call
applied on 3 rows. so,
it is called "multi row function"

greatest() one function call
applied on one row. so,
it is called "single row function"



String functions
Conversion functions
Date Functions
Analytic Functions
Number Functions

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

Min()	• is used to find min value in vertical values [column] • Syntax: min(<column>) • can take 1 argument.
--------------	--

	<ul style="list-style-type: none"> • it is multi row function
Least()	<ul style="list-style-type: none"> • is used to find min value in horizontal values [row] • Syntax: Least(<Arg1>, <arg2>,, <arg_n>) • can take variable length arguments • it is single row function.

Examples:

SELECT greatest(10,20,30), least(10,20,30) FROM dual;

Output:

30 10

SELECT max(Sal), min(Sal) FROM emp;

UID:

- returns user id

Syntax:

UID

User:

- it returns current user name

Syntax:

User

To see current user name:

SHOW USER

(or)

SELECT user FROM dual;

Display current user id and user name:

SELECT uid, user FROM dual;

Built-In Functions

String Functions	lower() upper() initcap() Lpad() Rpad() Ltrim() Rtrim() Trim() Replace() Translate()
Conversion	to_char() to_date() to_number()
Aggregate / Group / Multi Row	max() min() count() sum() avg()
Date	Add_Months() sysdate systimestamp Months_Between() Last_day() Next_day()
Analytic	Rank() Dense_Rank() Row_Number()
Number	trunc() ceil() floor() round() mod()
Other	NVL() NVL2() greatest() least() uid user

CLAUSES OF SELECT COMMAND

Saturday, March 30, 2024 10:11 AM

SQL QUERIES CLAUSES

ENGLISH SENTENCES WORDS

JAVA PROGRAMS INSTRUCTIONS

Query / Command

```
UPDATE emp  
SET sal=sal+1000  
WHERE job='MANAGER';
```

UPDATE clause
SET clause
WHERE clause

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

INSERT clause
INTO clause
VALUES clause

CLAUSES:

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

CLAUSES OF SELECT COMMAND:

SELECT command clauses are:

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

Syntax of SELECT command:

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

Display emp names and salaries of all managers:

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

SELECT	<ul style="list-style-type: none">• is used to specify column list• Example: SELECT ename, sal
FROM	<ul style="list-style-type: none">• is used to specify table names• Examples: FROM emp FROM emp, dept
WHERE	<ul style="list-style-type: none">• is used to specify filter condition• WHERE condition will be applied on every row• Examples: WHERE job='MANAGER' WHERE deptno IN(10,30)

ORDER BY:

- it is used to arrange the records in ascending or descending order according to specific column(s).

- **default order is: ASC**

Syntax:

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

Examples on ORDER BY:

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

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

(or)

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

(or)

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

ename	1
sal	2

*	empno, ename, job, mgr, hiredate, sal, comm, deptno						
1	2						

(or)

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

(or)

SELECT **empno, deptno, ename, sal**
FROM **emp**
ORDER BY **3;**

Display all emp records. arrange them in descending order according to salary:

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

(or)

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

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

break on deptno skip 1 duplicates

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

**Display all emp records.
arrange salaries in descending order within dept:**

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

**case-1: deptnos are different [if deptnos are different it will not check salary]
deptno**

..

deptno

20

10

10

20

case-2: deptnos are same [in this case only checks salary]

deptno sal

10

10

5000

8000

ORDER BY deptno ASC, sal DESC

10 8000

10 5000

**Display all emp records. arrange salaries
in descending order with in job:**

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

Display all emp records.
Arrange salaries in descending order with in dept. if salary is
same. arrange them according to seniority:

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

Case-1: deptnos are different

If deptnos are different it will not check salary or hiredate.

20 10
10 20

Case-2: deptnos are same.

if deptnos are same then only it checks salary.

10	5000	10	7000
10	7000	10	5000

Case-3: deptno and salary are same

if deptnos are same. then it checks salary. if salary is also same then only it checks hiredate.

10	5000	25-DEC-1983	10	5000	17-AUG-1980
10	5000	17-AUG-1980	10	5000	25-DEC-1983

sal	ORDER By sal DESC	ORDER BY sal ASC
-----	-----	-----
6000	null	4000
8000	null	6000
null	8000	7000
7000	7000	8000
4000	6000	null
null	4000	null

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

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

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

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

DISTINCT:

It is used to avoid [eliminate] duplicate rows.

Examples on DISTINCT:

Display the job titles offered by company:

```
SELECT job FROM emp;
```

(or)

```
SELECT ALL job FROM emp;
```

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

job

CLERK
MANAGER
MANAGER
CLERK
CLERK
MANAGER
SALESMAN
SALESMAN
CLERK
SALESMAN

job

CLERK
MANAGER
SALESMAN

Display the deptnos of company:

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

deptno

30
10
10
20
30
30
20
20
10
20
30

SELECT DISTINCT deptno
FROM emp
ORDER BY deptno;

deptno

10
20
30

Display the dept wise, job titles offered by company:

SELECT DISTINCT deptno,job FROM emp;

now DISTINCT will be applied on 2 columns

SMITH	10	CLERK
ALLEN	10	CLERK
WARD	10	CLERK
A	10	MANAGER
B	10	MANAGER
C	30	CLERK
D	30	CLERK

10 CLERK
10 MANAGER
30 CLERK

Example:

EMPLOYEE

EMPNO	ENAME	SAL
1001	A	6000
1002	B	5000
1003	C	8000
1001	A	6000
1002	B	5000
1003	C	8000

eliminate the duplicates from employee table:

**SELECT DISTINCT empno,ename,sal
FROM employee;**

GROUP BY:

- is used to group the records according to specific column(s).
- On these groups we can apply aggregate functions [group functions].
- It is used to get summarized data from detailed data [table].
- It is used for data analysis.

Example:

EMPLOYEE [detailed data]				GROUP BY deptno	summarized data	
EMPNO	ENAME	DEPTNO	SAL		DEPTNO	SUM_OF_SAL
1001	A	10	5000	sum()	10	25000
1002	B	10	20000		20	18000
1003	C	20	10000	sum()	30	22000
1004	D	20	8000			
1005	E	30	10000	sum()		
1006	F	30	12000			

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;
```

Execution Order of Clauses:

```
FROM
WHERE
GROUP BY
HAVING
SELECT [SELECT ALL]
DISTINCT
ORDER BY
OFFSET
FETCH
```

Display 10 and 30 depts sum of salaries:

```
SELECT deptno, sum(sal) AS sum_of_sal
FROM emp
WHERE deptno IN(10,30)
GROUP BY deptno
ORDER BY deptno;
```

EMP

EMPNO	ENAME	DEPTNO	SAL
1003	C	20	10000
1004	D	20	8000
1005	E	30	10000
1006	F	30	12000
1001	A	10	5000
1002	B	10	20000

FROM emp:

entire emp table will be selected

EMP

EMPNO	ENAME	DEPTNO	SAL
1003	C	20	10000
1004	D	20	8000
1005	E	30	10000
1006	F	30	12000
1001	A	10	5000
1002	B	10	20000

WHERE deptno IN(10,30):

it filters the rows. WHERE condition will be applied on every row

EMP

EMPNO	ENAME	DEPTNO	SAL
1003	C	20	10000
1004	D	20	8000
1005	E	30	10000
1006	F	30	12000
1001	A	10	5000
1002	B	10	20000

1005	E	30	10000
1006	F	30	12000
1001	A	10	5000
1002	B	10	20000

GROUP BY deptno:

it groups the records according to deptno.

if deptno is same that is one group

aggregate function will be applied on groups

1005	E	30	10000
1006	F	30	12000

30 group
sum(Sal) => 22000

1001	A	10	5000
1002	B	10	20000

10 group
sum(Sal) => 25000

SELECT deptno, sum(sal) AS sum_of_sal:

it selects specified columns

DEPTNO	SUM_OF_SAL
30	22000
10	25000

ORDER BY deptno:

it arranges records in ascending order

according to deptno

DEPTNO	SUM_OF_SAL
10	25000
30	22000

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 1;

```

Assignment:

SALES

DATEID	AMOUNT
1-JAN-2020	25000
2-JAN-2020	15000
..	
..	
2-APR-2024	30000

Find year wise sales:

YEAR	AMOUNT
2020	?
2021	?
..	

```

GROUP BY
to_char(dateid,'YYYY')

sum(amount)

```

Find quarter wise sales:

QUARTER	AMOUNT
1	?
2	?
3	?
4	?

```

?
GROUP BY
to_char(dateid,'Q')

sum(amount)

```

Assignment:

STUDENT

SID	SNAME	CNAME	FEE
1001	A	JAVA	5000
002	B	JAVA	5000
1003	C	JAVA	5000
1004	D	PYTHON	8000
1005	E	PYTHON	8000

Find course wise no of students

CNAME	NO_OF_STUDENTS
JAVA	?
PYTHON	?

```

GROUP BY cname
count(*)

```

1004	D	PYTHON	8000
1005	E	PYTHON	8000
1006	F	PYTHON	8000

GROUP BY cname
count(*)

Find course wise collected amount:

CNAME	AMOUNT
JAVA	?
PYTHON	?

GROUP BY cname
sum(fee)

Assignment:

PERSON

PID	PNAME	STATE	GENDER	AADHAR
123456	A	TS	M	
123457	B	TS	M	
	C	TS	F	
	D	TS	F	
	E	AP	M	
		AP	M	
		AP	F	
		AP	F	

find state wise population:

state	no_of_people
TS	?
AP	?

GROUP BY state
count(*)

find gender wise population:

GENDER	no_of_people
M	?
F	?

Find job wise sum of salaries:

JOB	SUM_OF_SAL
CLERK	?
MANAGER	?

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

Find job wise max salary and min salary:

JOB	MAX_SAL	MIN_SAL
CLERK	?	?
MANAGER	?	?

```
SELECT job, max(sal) AS max_sal, min(sal) AS min_sal
FROM emp
GROUP BY job;
```

Grouping the 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;
```

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 Year wise, Quarter wise no of emps:

YEAR	QUARTER	NO_OF_EMPS
1980	1	?
1980	2	?
1980	3	?
1980	4	?
1981	1	?
1981	2	?
1981	3	?
1981	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;
```

Assignment:

PERSON

PID	PNAME	STATE	GENDER	AADHAR
123456	A	TS	M	
123457	B	TS	M	
	C	TS	F	
	D	TS	F	
	E	AP	M	
		AP	M	
		AP	F	
		AP	F	

Find state wise, gender wise population:

STATE	GENDER	NO_OF_PEOPLE
TS	M	?
	F	?
AP	M	?
	F	?

Rollup() and Cube():

- These 2 functions are used to calculate sub totals and grand total.
- These 2 functions are called from GROUP BY clause.

Rollup():

Syntax:

GROUP BY Rollup(<grouping_column_list>)

Example:

GROUP BY Rollup(deptno, job)

Cube():

Syntax:

GROUP BY Cube(<grouping_column_list>)

Example:

GROUP BY Cube(deptno, job)

Examples on Rollup() and Cube():

Display dept wise, job wise no of emps.

Calculate sub totals and grand total according to deptno.

[Rollup()]:

DEPTNO	JOB	NO_OF_EMPS
10	CLERK	?
	MANAGER	?
	10th dept sub total	?
20	CLERK	?
	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;

Display 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	?
	MANAGER	?

	10th dept sub total	?
20	CLERK	?
	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.
Calculate sub totals and grand total according to deptno
[Rollup()]:

DEPTNO	JOB	SUM_OF_SAL
10	CLERK	?
	MANAGER	?
	10th dept sub total	?
20	CLERK	?
	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	?
	MANAGER	?
	10th dept sub total	?
20	CLERK	?
	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;**

**Display year wise, quarter wise no of emps
joined in organization.
Calculate sub totals and grand total according to
year:
[Rollup()]:**

YEAR	QTR	NO_OF_EMPS
1980	1	?
	2	?
	3	?
	4	?
	1980 sub total	?
1981	1	?

	2	?
	3	?
	4	?
	1981 sub total	?
	GRAND TOTAL	?

break on year skip 1

```
SELECT to_char(hiredate,'YYYY') AS year,
to_char(hiredate,'Q') AS quarter,
count(*) AS no_of_emps
FROM emp
GROUP BY Rollup(to_char(hiredate,'YYYY'), to_char(hiredate,'Q'))
ORDER BY 1;
```

**Display year wise, quarter wise no of emps
joined in organization.
Calculate sub totals and grand total according to
year and quarter:
[CUBE()]:**

YEAR	QRTR	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:

PERSON

PID	PNAME	STATE	GENDER	AADHAR
123456	A	TS	M	
123457	B	TS	M	
	C	TS	F	
	D	TS	F	
	E	AP	M	
		AP	M	
		AP	F	
		AP	F	

**Display state wise gender wise no of people.
Calculate sub totals and grand total according to state
[Rollup()]**

STATE	GENDER	NO_OF_PEOPLE
TS	M	?
	F	?
	TS population	?
AP	M	?
	F	?

	AP population	?
	INDIA population	?

**Display state wise gender wise no of people.
Calculate sub totals and grand total according to state and gender
[Cube()]**

STATE	GENDER	NO_OF_PEOPLE
TS	M	?
	F	?
	TS population	?
AP	M	?
	F	?
	AP population	?
	M population	?
	F population	?
	INDIA population	?

SALES

DATEID	AMOUNT
1-JAN-2020	25000
2-JAN-2020	15000
..	
..	
2-APR-2024	30000

**Display year wise, quarter wise salaes.
calculate sub totals and grand total
according to year [Rollup()]**

YEAR	QRTR	AMOUNT
2020	1	?
	2	?

	3	?
	4	?
	2020 sales	?
2021	1	?
	2	?
	3	?
	4	?
	2021 sales	?
	Grad total	?

**Display year wise, quarter wise salaes.
calculate sub totals and grand total
according to year and qrtr [Cube()]**

YEAR	QRTR	AMOUNT
2020	1	?
	2	?
	3	?
	4	?
	2020 sales	?
2021	1	?
	2	?
	3	?
	4	?
	2021 sales	?
	1st qrtr sales	?
	2nd qrtr sales	?
	3rd qrtr sales	?
	4th qrtr sales	?
	Grad total	?

HAVING clause:

- **Is used to write conditions on groups.**
- **It filters groups.**
- **it will be applied on result of GROUP BY.**
- **it cannot be used without GROUP BY.**

Examples on HAVING:

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

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

Display the depts which are having 5 or more emps:

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

Differences between WHERE and HAVING:

WHERE	HAVING
<ul style="list-style-type: none"> • WHERE clause condition will be applied on every row • it filters the rows • We cannot use Aggregate function in WHERE clause. • It gets executed before GROUP BY • it can be used without GROUP BY 	<ul style="list-style-type: none"> • HAVING clause condition will be applied on every group. • it filters the groups • We can use Aggregate function in HAVING clause • It gets executed after GROUP BY • it cannot be used without GROUP BY

What is the execution order of Clauses of SELECT command?

FROM
WHERE
GROUP BY
HAVING
SELECT [SELECT ALL]
DISTINCT
ORDER BY
OFFSET
FETCH

Can we use Column Alias in GROUP BY?

NO. [till oracle 21C]

Because, GROUP BY gets executed before SELECT

Find year wise no of emps joined in organization:

SELECT to_char(hiredate,'YYYY') AS year,
count(*) AS no_of_emps

FROM emp
GROUP BY year
ORDER BY year;

Output:

"YEAR" Invalid Identifier

Can we column alias in ORDER BY?

YES.

Because, ORDER BY gets executed after SELECT

Find year wise no of emps joined in organization:

SELECT to_char(hiredate,'YYYY') AS year,
count(*) AS no_of_emps
FROM emp
GROUP BY to_char(hiredate,'YYYY')
ORDER BY year;

Note:

- **From ORACLE 23c version onwards, we can use column alias in GROUP BY, HAVING**

Execution Order
[oracle 21c]

FROM
WHERE
GROUP BY
HAVING
SELECT
DISTINCT
ORDER BY
OFFSET
FETCH

Execution Order
[oracle 23c]

FROM
WHERE
SELECT
GROUP BY
HAVING
DISTINCT
ORDER BY
OFFSET
FETCH

OFFSET clause:

- introduced in **ORACLE 12C** version
- it is used to specify no of rows to be skipped

Syntax:

OFFSET <number> ROW/ROWS

FETCH clause:

- introduced in **ORACLE 12C** version
- it is used to specify no of rows to be fetched [selected]

Syntax:

FETCH FIRST/NEXT <number> ROW/ROWS ONLY

Examples on OFFSET and FETCH:

Display all emp records except first 5 rows:

```
SELECT * FROM emp  
OFFSET 5 ROWS;
```

Display first 5 rows only:

```
SELECT * FROM emp  
FETCH FIRST 5 ROWS ONLY;
```

Display top 3 salaried emp records:

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

Display top 3 seniors records:

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

Display 6th row to 10th row:

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

Execution Order [oracle 21c]:

FROM	used to specify table names	FROM emp
WHERE	<ul style="list-style-type: none">•used to specify condition•filters rows	WHERE sal>3000
GROUP BY	used to group the records according to specific column(s)	GROUP BY deptno
HAVING	<ul style="list-style-type: none">•used to specify condition•filters groups	HAVING count(*)>5
SELECT	used to specify column list	SELECT ename, sal
DISTINCT	used to avoid duplicate rows	SELECT DISTINCT job
ORDER BY	to arrange records in ASC or DESC	ORDER BY sal DESC
OFFSET	to skip the rows	OFFSET 5 ROWS
FETCH	to fetch the rows	FETCH FIRST 5 ROWS ONLY

JOINS

Thursday, April 4, 2024 9:22 AM

JOINS:

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

COLLEGE DB

STUDENT
MARKS
FEE
STAFF

•
•

S.SID = M.SID => Join Condition

STUDENT S

SID	SNAME	SCITY
1001 A		HYD
1002 B		MUMBAI
1003 C		DELHI
1004 D		PUNE

MARKS M

SID	MATHS	PHY	CHE
1001 70		90	80
1002 66		44	91
1003 77		40	80

JOINS

SID	SNAME	MATHS
STUDENT	STUDENT	MARKS

JOINS:

- JOIN => Combine / Connect / Link
- JOIN is an operation like Filtering, Sorting.
- In JOIN OPERATION, one table record will be joined with another table record based on condition. This condition is called Join Condition.
- 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
 - Equi Join
 - Non-Equi Join
- Outer Join
 - Left Outer Join
 - Right Outer Join
 - Full Outer Join

- Self Join
- Cross Join

Inner Join:

Inner Join gives matched records only.

It has 2 sub types. They are:

- Equi Join
- Non-Equi Join

Equi Join:

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

Example:

WHERE S.SID = M.SID

Example on Equi Join:

S.SID = M.SID

STUDENT S

SID	SNAME	SCITY
1001	A	HYD
1002	B	MUMBAI
1003	C	DELHI
1004	D	PUNE

MARKS M

SID	MATHS	PHY	CHE
1001	70	90	80
1002	66	44	91
1003	77	40	80

unmatched record from student table

CREATE TABLE student

```
(
sid NUMBER(4),
sname VARCHAR2(10),
scity VARCHAR2(10)
);
```

```
INSERT INTO student VALUES(1001,'A','HYD');
INSERT INTO student VALUES(1002,'B','MUMBAI');
INSERT INTO student VALUES(1003,'C','DELHI');
INSERT INTO student VALUES(1004,'D','PUNE');
COMMIT;
```

CREATE TABLE marks

```
(
sid NUMBER(4),
maths NUMBER(3),
phy NUMBER(3),
che NUMBER(3)
);
```

```
INSERT INTO marks VALUES(1001,70,90,80);
INSERT INTO marks VALUES(1002,66,44,91);
INSERT INTO marks VALUES(1003,77,40,80);
COMMIT;
```

Display student details with maths subject marks:

SID SNAME MATHS
STUDENT MARKS

SELECT SID, SNAME, MATHS
FROM student, marks
WHERE SID = SID;

Output:

ERROR: column ambiguously defined

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

Output:

ERROR: column ambiguously defined

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

Output:

SID	SNAME	MATHS
1001	A	70
1002	B	66
1003	C	77

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

Output:

SID	SNAME	MATHS
1001	A	70
1002	B	66
1003	C	77

Note:

above query degrades performance

To improve performance write above query as following:

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

Output:

SID	SNAME	MATHS
1001	A	70
1002	B	66
1003	C	77

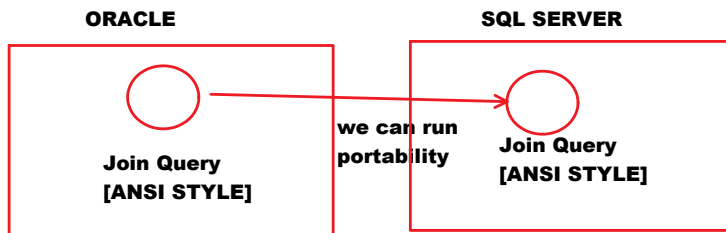
Note:

- From ORACLE 9i version onwards we can write Join Query in 2 styles. They are:
 - ORACLE STYLE / NATIVE STYLE
 - ANSI STYLE => Best way => **portability**

ORACLE

SQL SERVER

○ ANSI STYLE -- Best way -- portability



Note:

- In ORACLE STYLE, to separate 2 table names we write , [comma].
- In ANSI STYLE, to separate 2 table names we use KEYWORD
- In ORACLE STYLE, we write Join 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;
```

Examples on Equi Join:

e.deptno = d.deptno => Join Condition

EMP e				DEPT d		
EMPNO	ENAME	SAL	DEPTNO	DEPTNO	DNAME	LOC
7369	SMITH	800	20	10	ACCOUNTS	NEW YORK
7499	ALLEN	1600	30	20	RESEARCH	DALLAS
7521	WARD	1250	30	30	SALES	CHICAGO
7566	JONES	2975	20	40	OPERATIONS	BOSTON
7782	CLARK	2450	10			
7934	MILLER	1300	10			
1001	A	1800				
1002	B	2000				

Display the emp details along with dept details:

ENAME SAL DNAME LOC
EMP e DEPT d

ORACLE STYLE:

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

ANSI STYLE:

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

Display the emp records along with dept details.
display who are working in NEW YORK only:

ename	sal	dname	loc
			NEW YORK

Note:
to see execution plan write following command:
SQL> SET AUTOTRACE ON EXPLAIN

ORACLE STYLE:

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

ANSI STYLE:

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

Note:
ON clause is used to specify **JOIN CONDITION**
WHERE clause is used to specify **FILTER CONDITION**

First Filter condition gets executed then
Join Operation will be performed based on **JOIN CONDITION**.

e.deptno=d.deptno

EMP e				DEPT d		
EMPNO	ENAME	SAL	DEPTNO	DEPTNO	DNAME	LOC
7369	SMITH	800	20	10	ACCOUNTS	NEW YORK
7499	ALLEN	1600	30	20	RESEARCH	DALLAS
7521	WARD	1250	30	30	SALES	CHICAGO
7566	JONES	2975	20	40	OPERATIONS	BOSTON
7782	CLARK	2450	10			
7934	MILLER	1300	10			
1001	A	1800				
1002	B	2000				

Display ALLEN record along with dept details:

ename	sal	dname	loc
ALLEN			

ORACLE STYLE:

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

ANSI STYLE:

```
SELECT e.ename,e.sal,d.dname,d.loc
```



```

FROM emp e INNER JOIN dept d
ON e.deptno=d.deptno
WHERE e.ename='ALLEN';

```

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

Example on retrieving data from 4 tables:

e.deptno = d.deptno

d.locid=L.locid

L.cid=C.cid

EMP1 e

EMPNO
ENAME
DEPTNO

DEPT1 d

DEPTNO
DNAME
LOCID

LOCATION1 L

LOCID
LNAME
CID

COUNTRY1 c

CID
CNAME

ENAME	DNAME	LNAME	CNAME
emp1 e	dept1 d	location1 L	country1 c

Note:
 to retrieve data from 2 tables write 1 Join condition
 to retrieve data from 4 tables write 3 Join Conditions
 to retrieve data from 10 tables write 9 join conditions

n

n-1

ORACLE STYLE:

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;

```

SQL> select * from emp1;

EMPNO	ENAME	DEPTNO
5001	A	10

SQL> select * from dept1;

DEPTNO	DNAME	LOCID
10	SALES	101

SQL> select * from location1;

LOCID	LNAME	CID
101	HYD	1001

SQL> select * from country1;

CID	CNAME
1001	INDIA

Equi Join:

If Join Operation performed based on equality condition then it is called "Equi Join".

Examples:

WHERE e.deptno = d.deptno

Non-Equi Join:

If Join Operation performed based on other than equality condition then it is called "Non-Equi Join".

Examples:

WHERE e.deptno > d.deptno

WHERE e.deptno < d.deptno

WHERE e.deptno != d.deptno

Example on Non-Equi Join:

e.sal BETWEEN s.LOSAL AND s.HISAL

EMP e

EMPNO	ENAME	SAL
1001	A	2500
1002	B	6000
1003	C	1200
1004	D	1800
1005	E	3500
1006	F	

SALGRADE s

GRADE	LOSAL	HISAL
1	700	1400
2	1401	2000
3	2001	3000
4	3001	4000
5	4001	9999

ENAME SAL GRADE

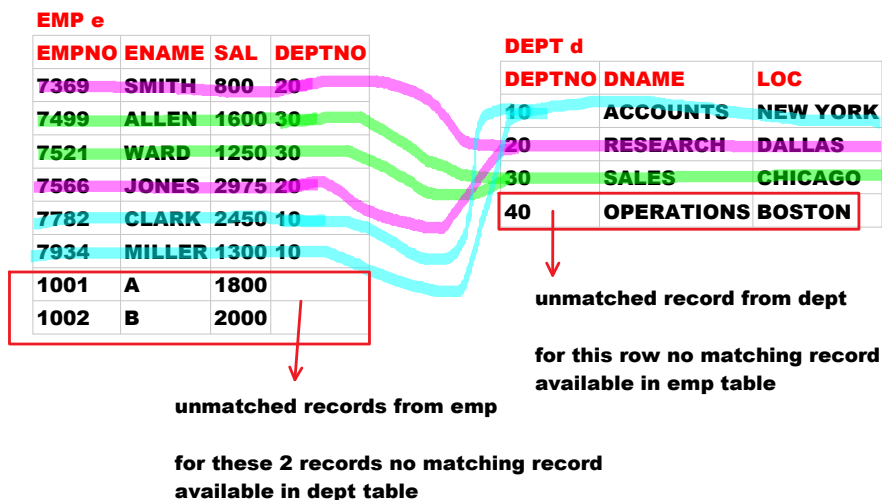
emp e salgrade s

ORACLE STYLE:

```
SELECT e.ename, e.sal, s.grade
FROM emp e, salgrade s
WHERE e.sal BETWEEN s.LOSAL AND s.HISAL;
```

ANSI STYLE:

```
SELECT e.ename, e.sal, s.grade
FROM emp e INNER JOIN salgrade s
ON e.sal BETWEEN s.losal AND s.hisal;
```



Outer Join:

- INNER JOIN can give matched records only.
To get unmatched records also we use OUTER JOIN.
- Inner Join = matched records only
- Outer Join = matched + unmatched records

There are 3 sub types in Outer Join:

- Left Outer Join = matched + unmatched from left table
- Right Outer Join
- Full Outer Join

Note:

In ORACLE STYLE, based on join condition we can decide left table and right table.

Examples:

WHERE e.deptno = d.deptno

emp e	left table
dept d	right table

WHERE d.deptno = e.deptno

dept d	left table
emp e	right table

In ANSI STYLE, **based on keyword** we can decide left table and right table

Examples:

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 gives matched records and unmatched records from left table.
- In ORACLE STYLE, write outer join operator (+) at right side.
- In ANSI style use the keyword: LEFT [OUTER] JOIN

Example on Left Outer Join:

```
INSERT INTO emp(empno,ename,sal)
VALUES(1001,'A',5000);
```

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

Display all emp details along with dept details as following.
Also display the emps to whom dept is not assigned

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 LEFT OUTER JOIN dept d
ON e.deptno=d.deptno;
```

Right Outer Join:

- **Right Outer Join = matched + unmatched from right table**

- **RIGHT OUTER JOIN** gives matched records and unmatched records from right table.
- In **ORACLE STYLE**, we write outer join operator (+) at left side.
- In **ANSI STYLE**, we use the keyword: **RIGHT [OUTER] JOIN**

Example on Right Outer Join:

Display all emp records along with dept details as following.
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**, for full outer join we use **UNION** operator between left outer join and right outer join. **UNION** operator combines result of both select queries without duplicates.

For Left Outer join	WHERE e.deptno = d.deptno(+)
For Right Outer Join	WHERE e.deptno(+)=d.deptno
For Full Outer Join	WHERE e.deptno(+) = d.deptno(+) ERROR
For Full Outer Join	Left Outer Join UNION Right Outer join

A = {1,2,3,4,5}
B = {4,5,6,7,8}

A U B = {1,2,3,4,5,6,7,8}

Left outer join
UNION
Right outer join

= matched + unmatched from left

= matched + unmatched from right

Full outer Join = Left Join UNION Right Join
matched + um from left + um from right

- In **ANSI STYLE**, we use the keyword: **FULL [OUTER] JOIN**

Example on Full Outer Join:

Display emp details along with dept details as following.
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:

Left Outer Join + Condition:
Left Outer Join + Condition gives only 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
-------	-----	-------	-----

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 Join + Condition:

Right Outer join Join + Condition gives only unmatched records from right table

Example on Right outer + Cond:

Display the depts which are not having emps as following:

ename	sal	dname	loc
SMITH	5000	RESEARCH	DALLAS
		OPERATIONS	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 + Conditions:

Full Outer Join + Conditions can give only unmatched records from left and right tables.

Example:

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

ename	sal	dname	loc
-------	-----	-------	-----

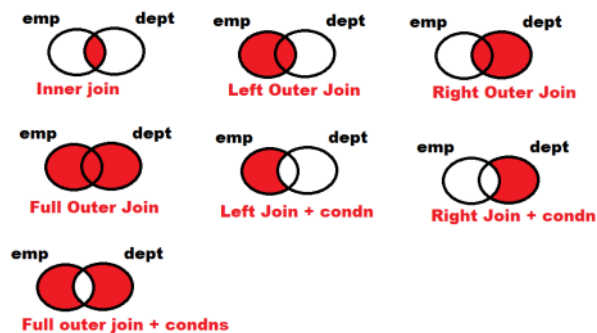
ORACLE STYLE:

```
SELECT e.ename, e.sal, d.dname, d.loc
FROM emp e, dept d
WHERE e.deptno=d.deptno(+) 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.
- Self Join can be also called as "Recursive Join".

Example on Self Join:

e.mgr = m.empno

EMP e

EMPNO	ENAME	JOB	SAL	MGR
1001	A	MANAGER	15000	
1002	B	CLERK	8000	1001
1003	C	ANALYST	6000	1001
1004	D	MANAGER	20000	
1005	E	CLERK	10000	1004
1006	F	SALESMAN	9000	1004

EMP m

EMPNO	ENAME	JOB	SAL	MGR
1001	A	MANAGER	15000	
1002	B	CLERK	8000	1001
1003	C	ANALYST	6000	1001
1004	D	MANAGER	20000	
1005	E	CLERK	10000	1004
1006	F	SALESMAN	9000	1004

Display emp details along with manager details:

emp_name	emp_sal	mgr_name	mgr_sal
emp e	emp e	emp m	emp m

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 details along with manager details.
Display the emps who are reporting to BLAKE only:

ORACLE STYLE:

```
SELECT e.ename AS emp_ename, e.sal AS emp_sal,  
m.ename AS mgr_name, m.sal AS mgr_sal  
FROM emp e, emp m  
WHERE e.mgr=m.empno AND m.ename='BLAKE';
```

ANSI STYLE:

```
SELECT e.ename AS emp_ename, e.sal AS emp_sal,  
m.ename AS mgr_name, m.sal AS mgr_sal  
FROM emp e INNER JOIN emp m  
ON e.mgr=m.empno  
WHERE m.ename='BLAKE';
```

Display the emp records who are earning more than
their manager:

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;

```

Example: a.cid < b.cid

GROUPA a		GROUPA b	
CID	CNAME	CID	CNAME
10	IND	10	IND
20	AUS	20	AUS
30	WIN	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 a.cname || ' VS ' || b.cname
FROM groupA a, groupA b
WHERE a.cid < b.cid;

```

ANSI STYLE:

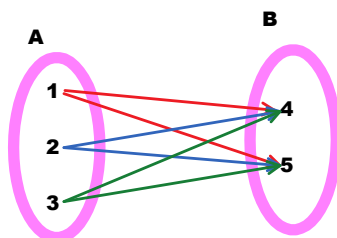
```

SELECT a.cname || ' VS ' || b.cname
FROM groupA a INNER JOIN groupA b
ON a.cid < b.cid;

```

Cross Join:

A = {1,2,3} AXB = {(1,4) (1,5) (2,4) (2,5) (3,4) (3,5)}
B = {4,5}
AXB = ?



- In Cross Join, Each record in one table will be joined

with every record in another table.

- For Cross Join we will not write any join condition.
- In ANSI style, we use the keyword: **CROSS JOIN**

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;
```

Assignment:

EMPLOYEE e

EMPNO	ENAME	PID
-------	-------	-----

PROJECTS p

PID	PNAME	DURATION
-----	-------	----------

Assignment:

EMPLOYEE e

EMPNO	ENAME	PID
1001	A	30
1002	B	30
1003	C	10
1004	D	10
1005	E	
1006	F	

PROJECTS p

PID	PNAME	DURATION
10	X	2
20	Y	1
30	Z	3

Display emp details with project details => Equi Join

ename	pname
employee e	projects p

Display emp details with project details as following.

Also display the emps to whom project is not assigned:

ename	pname
employee e	projects p

=> Left Outer Join

Display emp details with project details as following.

Also display the projects which are not assigned to any employee:

ename	pname
employee e	projects p

=> Right Outer Join

Display emp details with project details as following.

Also display the emps to whom project is not assigned.

Also display the projects which are not assigned to any employee.

ename	pname
employee e	projects p

=> Full Outer Join

display the emps to whom project is not assigned

as following:

ename	pname
employee e	projects p

=> Left Join + Cond'n

display the projects which are not assigned to any employee

as following:

ename	pname
employee e	projects p

=> Right Join + Cond'n

display the emps to whom project is not assigned.

display the projects which are not assigned to any employee

as following:

ename	pname
employee e	projects p

=> Full Join + Cond'n

JOINS:

- used to retrieve the data from multiple tables.
- JOIN => operation => combines one table record with another table record based on join condition

Types of Joins:

Inner Join		matched records only
	Equi Join	based on = join performed
	Non-Equi Join	based on other than = join performed
Outer Join		matched + unmatched
	Left outer	matched + um from left
	Right outer	matched + um from right
	Full Outer	matched + um from left and right
Self Join		a table will be joined to itself
Cross Join		each record in one table will be joined every record in another table

Sub Queries

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

Sub Queries

Monday, April 8, 2024 9:17 AM

Sub Queries / Nested Queries:

Syntax:

```
SELECT <column_list> / *  
FROM <table_name>  
WHERE <column> <operator> (<SELECT QUERY>);
```



**Outer Query /
Main Query /
Parent Query**

**Inner Query /
Sub Query /
Child Query**

Example:

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');
```

Sub Queries / Nested Queries:

- A query which is written in another query is called "Sub Query".
- Outside query is called "Outer Query / Main Query / Parent Query".
- Inside query is called "Inner Query / Sub Query / Child Query".

- When filter condition value is unknown to find it we write Sub Query.
- Sub Query must be written parenthesis.
- Sub Query must be **SELECT QUERY** only. It cannot be **INSERT / UPDATE / DELETE**. Because, Sub query has to find a value. Only **SELECT** can find the value.
- Main Query can be **SELECT / INSERT / UPDATE / DELETE**.
- In **WHERE** clause, we can write max of **254** sub queries. Including main query, we can write **255** levels of queries.
- First Inner query gets executed. Result of Inner Query becomes input for Outer query. Then Outer query gets executed.

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.
- Inner Query gets executed only **1 time**.

Correlated Sub Query:

- In Correlated Sub Query, **First Outer Query** gets executed. Then Inner query gets executed.
- Inner query gets executed for **multiple times**.

Single Row Sub Query:

- If sub query returns 1 value then it is called "Single Row Sub Query".

Examples on Single Row Sub Query:

Display the emp records who are earning more than BLAKE:

```
SELECT ename, sal
FROM emp
WHERE sal > (Find BLAKE sal);
```

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 same as ALLEN:

ALLEN	SALESMAN
-------	----------

```
SELECT ename, job, sal
FROM emp
WHERE job = (Find ALLEN job title);
```

Find ALLEN job title:
SELECT job FROM emp
WHERE ename='ALLEN';

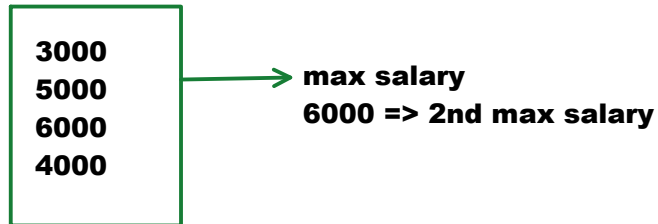
```
SELECT ename, job, sal
FROM emp
WHERE job = (SELECT job FROM emp
WHERE ename='ALLEN');
```

Find 2nd max salary:

**SELECT max(sal) FROM emp
WHERE sal<(Find max salary);**

sal

3000
7000
5000
6000
4000



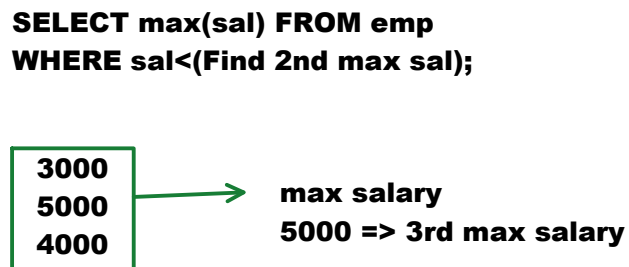
**Find max salary:
SELECT max(Sal) FROM emp;**

**SELECT max(Sal) FROM emp
WHERE sal<(SELECT max(sal)
FROM emp);**

Find 3rd max salary:

sal

3000
7000
5000
6000
4000



**SELECT max(Sal) FROM emp
WHERE sal<(SELECT max(sal) FROM emp
WHERE sal<(SELECT max(sal) FROM emp));**

**Note:
When we use Group Function [Aggregate Function],
SELECT clause allows GROUP BY column or GROUP
FUNCTION only.**

**SELECT ename, max(Sal)
FROM emp;**

Output:

**ERROR: ename is NOT GROUP BY column.
ename is NOT GROUP FUNCTION**

**When we use GROUP BY,
SELECT clause allows GROUP BY column or GROUP
FUNCTION only.**

**SELECT deptno, ename, max(Sal)
FROM emp
GROUP BY deptno;**

Output:

**ERROR:
ename is NOT GROUP BY column
ename is NOT GROUP FUNCTION**

Find the emp name 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 earning 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));**

Assignment:

- **Find the emp name who is earning 3rd max sal**
- **Find the emp name who is earning min sal**

Display the emp records who are senior to BLAKE:

```
SELECT ename, hiredate  
FROM emp  
WHERE hiredate<(Find BLAKE hiredate);
```

```
SELECT ename, hiredate  
FROM emp  
WHERE hiredate<(SELECT hiredate FROM emp  
WHERE ename='BLAKE');
```

Display the emp records who are juniors of BLAKE:

```
SELECT ename, hiredate  
FROM emp  
WHERE hiredate>(Find BLAKE hiredate);
```

```
SELECT ename, hiredate  
FROM emp  
WHERE hiredate>(SELECT hiredate FROM emp  
WHERE ename='BLAKE');
```

Display most senior record:

```
SELECT *  
FROM emp  
WHERE hiredate=(find most senior hiredate);
```

```
SELECT *  
FROM emp  
WHERE hiredate=(SELECT min(hiredate) FROM emp);
```

Display most junior record:

```
SELECT *  
FROM emp  
WHERE hiredate=(find most junior hiredate);
```

```
SELECT *  
FROM emp  
WHERE hiredate=(SELECT max(hiredate) FROM emp);
```

Set 7900 salary as deptno 30's max sal:

deptno 30's max sal	2850
---------------------	------

```
UPDATE emp
SET sal=(Find deptno 30 max sal)
WHERE empno=7900;
```

Find deptno 30 max sal:

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

```
UPDATE emp
SET sal=(SELECT max(sal) FROM emp WHERE deptno=30)
WHERE empno=7900;
```

Delete most senior record:

```
DELETE FROM emp
WHERE hiredate=(Find most senior's hiredate);
```

Find most senior's hiredate:

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

```
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 sum of sal in all depts);
```

find max sum of sal in all depts:

```
SELECT max(sum(sal)) FROM emp
GROUP deptno;
```

deptno	sum(sal)
10	8000
20	10000
30	9000

```
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:

```
SELECT dname  
FROM dept  
WHERE deptno=(find the deptno which is spending max  
amount);
```

```
SELECT dname FROM dept  
WHERE deptno=(SELECT deptno FROM emp  
GROUP BY deptno  
HAVING sum(sal)=(SELECT max(sum(Sal)) FROM emp  
GROUP BY deptno));
```

Multi Row Sub Query:

- **If sub query returns multiple rows then it is called "Multi Row Sub Query".**
- **For multi row sub query we cannot use relational operators [= > < <= >= !=].**
We must use IN, ANY, ALL.

Examples on multi row sub query:

Display the emp records whose job title is same as BLAKE and SMITH:

```
SELECT ename, job, sal  
FROM emp  
WHERE job IN(Find job titles of BLAKE and SMITH);
```

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

**WHERE job IN(SELECT job FROM emp
WHERE ename IN('BLAKE','SMITH'));**

ALL:

Syntax:

<column> <relational_operator> ALL(<value_list>)

Example:

sal > ALL(2000,3000)

if sal is > all list of values then condn is TRUE

sal	sal > ALL(2000,3000)	sal>2000 AND sal>3000
5000	T	
2500	F	
1000	F	
4000	T	
2800	F	

- **ALL operator is used for multi value comparison.**
- **ALL operator is prefixed with relational operator.**
- **it avoids of writing multiple relational conditions using AND.**

ANY:

Syntax:

Example:

sal > ANY(2000,3000)

if sal > any one of list of values then condn is TRUE

sal > ANY(2000,3000) sal>2000 OR sal>3000

sal > ALL(2000,3000)

sal

5000	T
2500	F
1000	F
4000	T
2800	F

sal > ANY(2000,3000)

sal

5000	T
2500	T
1000	F
4000	T
2800	T

sal > ALL(2000,3000)	sal>2000 AND sal>3000
sal > ANY(2000,3000)	sal>2000 OR sal>3000

sal=2000 OR sal=3000	sal IN(2000,3000)	sal=ANY(2000,3000)
----------------------	-------------------	--------------------

IN	=ANY
----	------

Examples on ALL and ANY:

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'); => multi row sub query
```

```
SELECT ename, sal FROM emp
WHERE sal>(SELECT max(Sal) FROM emp
WHERE job='MANAGER'); => single row sub query
```

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'); => multi row sub query
```

```
SELECT ename, sal FROM emp
WHERE sal > (SELECT min(sal) FROM emp
WHERE job='MANAGER'); => single row sub query
```

Assignment:

Display the emp records who are earning more than all emps of deptno 20:

WHERE sal > ALL(find deptno 20 all sals)

Display the emp records who are earning more than any 1 of emps of deptno 20:

WHERE sal > ANY(find deptno 20 all sals)

Inline View / Inline Sub Query:

- if Sub Query is written in FROM clause then it is called "Inline View".
- Sub query acts like table.
- It is used to control execution order of clauses.

Syntax:

```
SELECT <column_list> / *
FROM (<SELECT query>)
WHERE <condition>;
```

Examples on Inline View:

Find 3rd max salary:

```
SELECT ename, sal,
dense_rank() over(order by sal desc) as rank
FROM emp
WHERE rank=3;
Output:
```

Execution Order:

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


FROM emp
WHERE rank=3;
Output:
ERROR: "RANK" invalid identifier

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 nth max salary:

SELECT DISTINCT sal
FROM (SELECT ename, sal,
dense_rank() over(order by sal desc) as rank
from emp)
where rank=&n;

Output:
Enter value for n: 3
old 5: where rank=&n
new 5: where rank=3

SAL

2975

Enter value for n: 5
old 5: where rank=&n
new 5: where rank=5

SAL

2450

Display top 3 salaried emp records:

SELECT *
FROM (SELECT ename, sal,
dense_rank() over(order by sal desc) as rank
FROM emp)

WHERE rank<=3;

Display 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:

ROWNUM
ROWID

Pseudo	False
---------------	--------------

ROWNUM:

- is a pseudo column
- is used to apply row numbers to records
- row numbers will be applied on result of **SELECT** query

Examples on ROWNUM:

Display all emp records. apply row numbers to them:

```
SELECT rownum AS sno, *  
FROM emp;  
Output:  
Error
```

```
SELECT rownum AS sno, e.*  
FROM emp e;
```

Display enames and salaries along with row numbers.
Display whose salaries are more than 2500 only:

```
SELECT rownum AS sno, ename, sal  
FROM emp  
WHERE sal>2500;
```

Examples on Inline View:

Display 3rd record from emp table:

```
SELECT *  
FROM (SELECT rownum as rn, ename, sal  
FROM emp)  
WHERE rn=3;
```

Display 3rd, 7th and 10th rows:

```
SELECT *  
FROM (SELECT rownum as rn, ename, sal  
FROM emp)  
WHERE rn IN(3,7,10);
```

Display 5th row to 10th row:

```
SELECT *  
FROM (SELECT rownum as rn, ename, sal  
FROM emp)  
WHERE rn BETWEEN 5 AND 10;
```

Display even numbered rows:

```
SELECT *  
FROM (SELECT rownum as rn, ename, sal  
FROM emp)  
WHERE Mod(rn,2)=0;
```

ROWID:

- **is a pseudo column.**
- **it is used to get physical address of row.**
- **to manipulate duplicate records it is useful.**

Examples:

Display all emp records along with rowids:

```
SELECT rowid, ename, sal
```

FROM emp;

Example:

EMPLOYEE			
EMPNO	ENAME	SAL	rowid
1001	A	5000	AAA
1001	A	5000	AAB

delete duplicated record from above table:

**DELETE FROM emp
WHERE rowid=<rowid>;**

**delete from employee
where rowid='AAAVOJAAHAAAAAtGAAB';**

Scalar Sub Query:

- If sub query is written in **SELECT** clause then it is called "Scalar Sub Query".
- It acts like column.

Syntax:

```
SELECT (<SELECT query>)  
FROM <table_name>  
WHERE <condition>;
```

Examples on Scalar Sub Query:

Display no of records in emp and dept tables:

```
SELECT (SELECT count(*) FROM emp) AS emp,  
(SELECT count(*) FROM dept) AS dept  
FROM dual;
```

Output:

EMP	DEPT
-----	-----
14	4

Calculate share of each department:

display output as following:

DEPTNO	SUM_OF_SAL	TOTAL_AMOUNT	PER
10	8750	29025	$8750 * 100 / 29025 = 30.1464$
20	10875	29025	$10875 * 100 / 29025 = 37.4677$
30	9400	29025	$9400 * 100 / 29025 = 32.3859$

```

SELECT deptno, sum(Sal) AS sum_of_sal,
(select sum(sal) from emp) AS total_amount,
trunc(sum(sal)*100/(select sum(Sal) FROM emp),2) AS per
FROM emp
GROUP BY deptno
ORDER BY 1

```

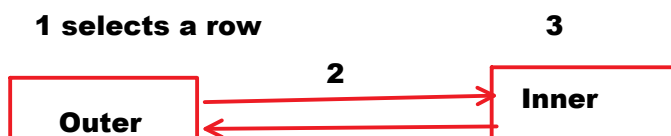
Non-Correlated Sub Query:

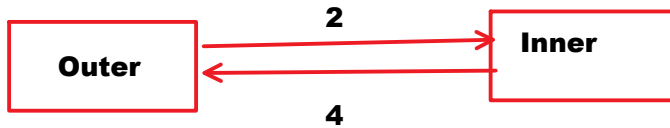
- First, inner query gets executed.
- Inner query gets executed only once.

Correlated Sub Query:

- First, outer query gets executed.
- Inner query gets executed for multiple times.
- If outer query passes value to inner query then it is called "Correlated Sub Query"

Execution Process of Correlated Sub Query:





5 Condition => T selects row

- 1. Outer Query gets executed. It selects a row.**
- 2. Outer query passes value to Inner Query.**
- 3. Inner Query gets executed.**
- 4. Inner query passes value to Outer query.**
- 5. Outer query condition will be tested. If condition is TRUE selects the row. Otherwise, row will not be selected.**

These 5 steps will be executed repeatedly for every row selected by Outer Query.

Example on Correlated Sub Query:

Display the emp records who are earning more than their dept's avrg salary:

EMP e

EMPNO	ENAME	DEPTNO	SAL
1001	A	10	10000
1002	B	20	20000
1003	C	30	30000
1004	D	10	15000
1005	E	20	10000
1006	F	30	10000

DEPTNO	AVG(SAL)
10	12500
20	15000
30	20000

sal>his dept's avrg sal

SELECT ename, deptno, sal
FROM emp e
WHERE sal>(select avg(Sal)
 from emp where deptno=e.deptno);

B	20	20000
C	30	30000
D	10	15000

Display the emp records who are earning maximum salary in each dept:

Emp Sal = emp dept's max sal

EMP e

EMPNO	ENAME	DEPTNO	SAL
1001	A	10	10000
1002	B	20	20000
1003	C	30	30000
1004	D	10	15000
1005	E	20	10000
1006	F	30	10000

```
SELECT ename, deptno, sal
FROM emp e
WHERE sal = (SELECT max(Sal)
FROM emp
WHERE deptno=e.deptno);
```

B	20	20000
C	30	30000
D	10	15000

Assignment:

Display most senior emp record in each dept:

hiredate = emp dept's most senior's hiredate

Exists:

Syntax:

Exists(<Sub Query>)

- **If sub query selects rows then rows are existed. So, it returns TRUE.**
- **If sub query does not select the rows then rows are not existed. So, it returns FALSE.**

Example:

Display the depts which are having emps:

DEPT d

DEPTNO	DNAME	LOC
10	ACCOUNTING	..
20	RESEARCH	..
30	SAL	..
40	OPERATIONS	..

EMP

EMPNO	ENAME	DEPTNO	SAL
1001	A	10	10000
1002	B	20	20000
1003	C	30	30000
1004	D	10	15000
1005	E	20	10000
1006	F	30	10000

```
SELECT * FROM dept d
WHERE EXISTS(SELECT * FROM emp
WHERE deptno=d.deptno);
```

10	ACCOUNTING	..
20	RESEARCH	..
30	SAL	..

Display the depts which are having emps:

```
SELECT * FROM dept d
WHERE NOT EXISTS(SELECT * FROM emp
WHERE deptno=d.deptno);
```

Sub Queries:

- A query which is written in another query
- Outside query => outer / main / parent
- Inside query => inner / sub / child

2 Types:

Non-Correlated		inner query get executed first inner query gets executed once			
	Single Row SQ	SQ returns 1 row			
	Multi Row SQ	SQ returns multiple rows			
	Inline View	If we Write SQ in FROM clause			
	Scalar	If we write SQ in SELECT clause			
Correlated		outer query gets executed first inner query gets executed multiple times			

		no of execution times of inner query = no of rows selected by outer query			
--	--	--	--	--	--

In WHERE clause, we can write max of 254 Sub Queries.

In FROM clause, we can write unlimited no of Sub Queries

In SELECT clause, we can write unlimited no of Sub Queries.

Constraints

Saturday, April 13, 2024 10:37 AM

Constraints:

Constraint => restrict / limit / control

**Max marks: 100
0 AND 100**

STUDENT CHECK(M1 BETWEEN 0 AND 100)

SID	SNAME	M1
1001	A	70
1002	B	56
1003	C	782 ERROR

Constraints:

- **Constraint is a rule that is applied on a column.**
- **Constraint is used to restrict the user from entering invalid data.**
- **it is used to maintain accurate and quality data.**
- **maintaining accurate and quality data is called "Data integrity".**
- **to implement data integrity feature we use Constraints [Integrity Constraints].**

ORACLE SQL provides following Constraints:

- **Primary Key**
- **Not Null**
- **Unique**
- **Check**

- **Default**
- **References [Foreign Key]**

Primary Key:

- It **does not duplicates**.
- It **does not accept nulls**.
- When value is mandatory and it must be unique then use PK.
- A table can have only 1 primary key.

Example:

EMP
PK

EMPNO	ENAME	JOB	SAL
1001	A	CLERK	5000
1002	B	CLERK	8000
1003	A	SALESMAN	5000
1001 => ERROR:duplicate	C	MANAGER	12000
=> ERROR: null	D	CLERK	6000

Example on Primary Key:

T1

F1	NUMBER(4)	PRIMARY KEY
-----------	------------------	--------------------

CREATE TABLE t1

(
f1 NUMBER(4) PRIMARY KEY
);

INSERT INTO t1 VALUES(1001);
INSERT INTO t1 VALUES(1002);

INSERT INTO t1 VALUES(1001);
Output:

ERROR: unique constraint violated

INSERT INTO t1 VALUES(null);

Output:

ERROR: cannot insert null into c##batch9am.T1.F1

Example:

T2

F1	don't accept duplicates and nulls	PK
F2	don't accept duplicates and nulls	unique not null

CREATE TABLE t2

**(
f1 NUMBER(4) PRIMARY KEY,
f2 VARCHAR2(10) PRIMARY KEY
);**

Output:

ERROR: A table can have 1 PK

CREATE TABLE t2

**(
f1 NUMBER(4) PRIMARY KEY,
f2 VARCHAR2(10) UNIQUE NOT NULL
);**

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
1001	A	5000
1002	B	8000
1003	A	5000
1004	ERROR: null	6000

Example on Not Null:

```
CREATE TABLE t3  
(  
f1 NUMBER(4) NOT NULL  
);
```

```
INSERT INTO t3 VALUES(1);  
INSERT INTO t3 VALUES(2);  
INSERT INTO t3 VALUES(1);
```

```
INSERT INTO t3 VALUES(null);
```

Output:

ERROR: cannot insert null into c##batch9am.T3.F1

UNIQUE:

- It does not accept duplicates.
- It accepts nulls.
- When value is optional and it must be unique then use **UNIQUE**.

Example:

CUSTOMER

UNIQUE

CID	CNAME	MAIL_ID
1001	raju	raju@gmail.com
1002	kiran	
1003	amar	raju@gmail.com ERROR: duplicate

Example:

```
CREATE TABLE t4
(
f1 NUMBER(4) UNIQUE
);
```

```
INSERT INTO t4 VALUES(1001);
INSERT INTO t4 VALUES(1002);
INSERT INTO t4 VALUES(null);
```

```
INSERT INTO t4 VALUES(1001);
```

Output:

ERROR: unique constraint violated

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.

Example:

STUDENT

CHECK(m1 BETWEEN 0 AND 100)

**max marks:100
0 to 100**

SID	SNAME	M1
1001	A	70
1002	B	89
1003	C	567 ERROR

GENDER CHECK(gender IN('M','F'))

M

M

F

F

M

Z => ERROR

Q => ERROR

DEFAULT:

- It is used to apply default value to column

Example:

STUDENT

DEFAULT 20000

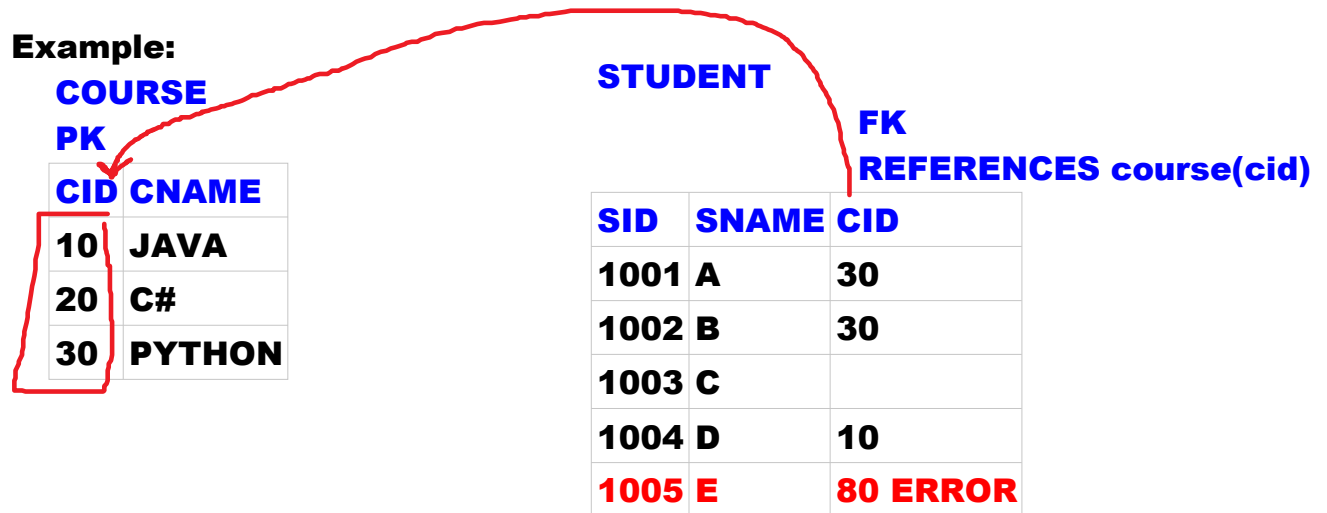
SID	SNAME	FEE
1001	A	20000
1002	B	20000
1003	C	20000

1004	D	10000
------	---	-------

REFERENCES [FOREIGN KEY]:

- **FOREIGN KEY** accepts **PRIMARY KEY** values of another table.
- To set Foreign Key use the keyword: **REFERENCES**
- **FK** can accept duplicates.
- **FK** can accept nulls.

Example:



Examples on Constraints:

STUDENT

SID	SNAME	M1
-----	-------	----

SID	don't accept duplicates and nulls	PK
SNAME	don't accept nulls	NOT NULL
m1	marks must be b/w 0 to 100	CHECK


```
CREATE TABLE student  
(  
sid NUMBER(4) PRIMARY KEY,  
sname VARCHAR2(10) NOT NULL,  
m1 NUMBER(3) CHECK(m1 BETWEEN 0 AND 100)  
);
```

```
INSERT INTO student VALUES(1,'A',70);
```

```
INSERT INTO student VALUES(1,'B',70);
```

Output:

ERROR: unique constraint violated

```
INSERT INTO student VALUES(null,'B',70);
```

Output:

ERROR: cannot insert NULL into student.sid

```
INSERT INTO student VALUES(2,'A',60);
```

```
INSERT INTO student VALUES(3,null,50);
```

Output:

ERROR: cannot insert NULL into student.sname

```
INSERT INTO student VALUES(4,'D',567);
```

Output:

ERROR: check constraint violated

Example-2:

Users_list

USER_ID	UNAME	PWD
1001	raju	kumar1234

UID	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(20) CHECK(length(pwd)>=8)
);
```

Example:

STUDENT

		NARESH	HYD	20000
SID	SNAME	CNAME	CCITY	FEE

sid	don't accept dups and nulls	PK
sname	don't accept nulls	NOT NULL
CNAME	default value NARESH	DEFAULT
CCITY	default value HYD	DEFAULT
FEE	default value 20000	DEFAULT

```
CREATE TABLE student
(
  sid NUMBER(4) PRIMARY KEY,
  sname VARCHAR2(10) NOT NULL,
  cname VARCHAR2(10) DEFAULT 'NARESH',
  ccity VARCHAR2(10) DEFAULT 'HYD',
  fee NUMBER(7,2) DEFAULT 20000
);
```

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

Output:

ERROR: not enough values

INSERT INTO student(sid,sname) VALUES(1001,'A');

INSERT INTO student(sid,sname,fee) VALUES(1002,'B',10000);

Example:

DEPT1 [master table]
PK

DEPTNO	DNAME
10	HR
20	SALES
30	ACCOUNTS

EMP1 [detailed table]
FK

PK	REFERENCES dept1(deptno)	
EMPNO	ENAME	DEPTNO
1001	A	30
1002	B	30
1003	C	10
1004	D	20
1005	E	
1006	F	90 ERROR

```
CREATE TABLE dept1  
(  
deptno NUMBER(2) PRIMARY KEY,  
dname VARCHAR2(10)  
);
```

```
CREATE TABLE emp1  
(  
empno NUMBER(4) PRIMARY KEY,  
ename VARCHAR2(10) NOT NULL,  
deptno NUMBER(2) REFERENCES dept1(deptno)  
);
```

Note:

PRIMARY KEY can be also called as **Parent Key**

FOREIGN KEY can be also called as **Child Key**

The table which has PK is called "Master table"

The table which has FK is called "Detailed table".

Assignment:

COURSE

PK

CID	CNAME
10	JAVA
20	ORACLE
30	PYTHON

STUDENT

FK

REFERENCES course(cid)

SID	SNAME	CID
1001	A	20
1002	B	20
1003	C	10
1004	D	30
1005	E	90 ERROR

Customers

PK

cid	cname	ccity
1001		
..		
1099		

PRODUCTS

PK

pid	pname	price
100		
..		
999		

ORDERS

FK FK

order_id	cid	pid	qty
123456	1001	105	

Syntax to create the table:

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

Naming Constraints

Tuesday, April 16, 2024 9:56 AM

Naming Constraints:

- we can give names to constraints.
- to identify constraint uniquely in ORACLE DB this name is required.
- when we define the constraint to a column it's better to give constraint name. If we don't give constraint name, implicitly oracle defines a constraint name.
This oracle defined constraint is: a six digit random number will be prefixed with SYS_C
Example:
SYS_C008798
- To drop the constraint or enable the constraint or disable the constraint this name is required.
- CONSTRAINT keyword is used to give constraint name

Syntax to create the table:

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

Example:

STUDENT

SID	SNAME	M1
-----	-------	----

PK	NOT NULL	CHECK [0 to 100]
c1	c2	c3 => con names

CREATE TABLE student

```
(  
sid NUMBER(4) CONSTRAINT c1 PRIMARY KEY,  
sname VARCHAR2(10) CONSTRAINT c2 NOT NULL,  
m1 NUMBER(3) CONSTRAINT c3 CHECK(m1 BETWEEN 0 AND 100)  
);
```

Note:

We cannot give name to DEFAULT constraint.

CREATE TABLE student

```
(  
sid NUMBER(4),  
sname VARCHAR2(10),  
m1 NUMBER(3),  
CONSTRAINT c1 PRIMARY KEY(sid)  
);
```


We can apply constraint at 2 levels. They are:

- **Column Level**
- **Table Level**

Column Level Constraint:

- **If constraint is defined in column definition then it is called "Column Level Constraint".**
- **all 6 constraints can be applied at column level.**
[PK, NOT NULL, UNIQUE, CHECK, DEFAULT, FK]

Example on Column Level Constraints:

```
CREATE TABLE student  
(  
sid NUMBER(4) CONSTRAINT c1 PRIMARY KEY,  
sname VARCHAR2(10) CONSTRAINT c2 NOT NULL,  
m1 NUMBER(3) CONSTRAINT c3 CHECK(m1 BETWEEN 0 AND 100)  
);
```

c1, c2, c3 constraints applied at column level.

Table Level Constraint:

- **If constraint is defined after defining all columns then it is called "Table Level Constraint".**
- **only 4 constraints can be applied at table level.**
they are:
PRIMARY KEY
UNIQUE
CHECK
FOREIGN KEY

Note:

- **NOT NULL** and **DEFAULT** constraints cannot be applied at table level.

Example on table level constraint:

STUDENT1

SID	SNAME	M1
PK con1	NOT NULL con2	CHECK con3

define **PK** and **CHECK** at table level.

define **NOT NULL** at column level.

```
CREATE TABLE student1
(
  sid NUMBER(4),
  sname VARCHAR2(10) CONSTRAINT con2 NOT NULL,
  m1 NUMBER(3),
  CONSTRAINT con1 UNIQUE(sid),
  CONSTRAINT con3 CHECK(m1 BETWEEN 0 AND 100)
);
```

Example:

COURSE2

PK

CID	CNAME
10	JAVA
20	C#
30	PYTHON

apply **PK** at table level

STUDENT2

FK

SID	SNAME	CID
1001	A	30
1002	B	10
1003	C	10
1004	D	80 ERROR

apply **FK** at table level

```
CREATE TABLE course2
```

```
(
cid NUMBER(2),
cname VARCHAR2(10),
CONSTRAINT con10 PRIMARY KEY(cid)
);

CREATE TABLE student2
(
sid NUMBER(4),
sname VARCHAR2(10),
cid NUMBER(2),
CONSTRAINT con11 FOREIGN KEY(cid) REFERENCES course2(cid)
);
```

Why Table Level?

2 reasons:

- **to apply constraint on combination of columns**
we use **Table level**.
- **to use another column name in constraint**
we use **table level**.

Note:

above 2 cases are possible at table level only.
not possible at column level.

Note:

If PK is applied on combination of columns then
it is called "Composite Primary Key**".**

Example:

STUDENT3
PK(SID,SUBJECT)

SID	SNAME	SUBJECT	MARKS
1001	A	m1	70
1001	A	m2	90
1001	A	m3	70
1002	B	m1	66
1002	B	m2	90
1002	B	m3	50
1001		m1	ERROR:duplicate
null	ERROR		
		null	ERROR

```

CREATE TABLE student3
(
  sid NUMBER(4),
  sname VARCHAR2(10),
  subject CHAR(2),
  marks NUMBER(3),
  CONSTRAINT con20 PRIMARY KEY(sid,subject)
);

```

Example:

CMS_LIST		CHECK(end_date>start_date)	
STATE_CODE	CM_NAME	START_DATE	END_DATE
TS	RR	9-DEC-2023	9-DEC-2020 ERROR

```

CREATE TABLE cms_list
(
  state_code CHAR(2),
  cm_name VARCHAR2(20),
  start_date DATE,
  end_date DATE CONSTRAINT con30 CHECK(end_date>start_date)
);

```

Altering Constraints

Wednesday, April 17, 2024 9:31 AM

ALTER:

- add the columns
- rename the columns
- drop the columns
- modify the data types
- modify the field sizes
- we can add the constraints
- rename the constraints
- disable the constraints
- enable the constraints
- drop the constraints

Syntax:

```
ALTER TABLE <table_name> [ADD CONSTRAINT <con_type>(<column>)]  
[RENAME CONSTRAINT <old_name> TO <new_name>]  
[DISABLE CONSTRAINT <con_name>]  
[ENABLE CONSTRAINT <con_name>]  
[DROP CONSTRAINT <con_name>];
```

Example:

STUDENT

SID	SNAME	M1
-----	-------	----

```
CREATE TABLE student  
(  
  sid NUMBER(4),  
  sname VARCHAR2(10),  
  m1 NUMBER(3)  
);
```

Add primary key to sid:

```
ALTER TABLE student ADD CONSTRAINT con50 PRIMARY KEY(sid);
```

Note:

- Using **ADD CONSTRAINT** keyword we can add Table level Constraints only [PK, UNIQUE, CHECK, FK]
- To add NOT NULL or DEFAULT constraints use **MODIFY** keyword
- Using **MODIFY** we can add all column level constraints [6 constraints]

Add not null to sname:

```
ALTER TABLE student MODIFY sname constraint con51 NOT NULL;
```

Add check constraint to m1:

```
ALTER TABLE student  
ADD CONSTRAINT con52 CHECK(m1 BETWEEN 0 AND 100);
```

Rename The constraint [con50 to z]:

```
ALTER TABLE student  
RENAME CONSTRAINT con50 TO z;
```

Disable the constraint z:

```
ALTER TABLE student DISABLE CONSTRAINT z;
```

Enable the constraint z:

```
ALTER TABLE student ENABLE CONSTRAINT z;
```

Drop the constraint z:

```
ALTER TABLE student DROP CONSTRAINT z;
```

How to see constraint names?

user_constraints:

- **it is a system table / readymade table.**
- **It maintains all constraints information.**

to see constraints list of a table:

```
SELECT table_name, constraint_name, constraint_type  
FROM user_constraints  
WHERE table_name='STUDENT';
```

SET OPERATORS

Wednesday, April 17, 2024 10:09 AM

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 U A B = {1,2,3,4,5,4,5,6,7,8} = B U A A

A I B = {4,5} = B I A

**A M B = {1,2,3} => specific elements of A
gives all elements from A except common elements**

A M B != B M A

B M A = {6,7,8} => specific elements of B

SET OPERATORS:

Syntax:

<SELECT QUERY>
<SET OPERATOR>
<SELECT QUERY>;

- **SET OPERATORS are used to combine result of 2 SELECT QUERIES.**
- **ORACLE SQL provides following SET OPERATORS.**
They are:
 - **UNION**

- **UNION ALL**
- **INTERSECT**
- **MINUS**

UNION:

- **it combines result of 2 select queries without duplicates**
- **it does not give duplicates**

UNION ALL:

- **it combines result of 2 select queries including duplicates**
- **it gives duplicates**

INTERSECT:

- **it given common records**

MINUS:

- **it gives specific records**

Example:

deptno 10

MANAGER
PRESIDENT
CLERK

deptno 20

CLERK
MANAGER
ANALYST

Display the job titles of deptno 10 and 20:

SELECT job FROM emp WHERE deptno=10
UNION
SELECT job FROM emp WHERE deptno=20;

Output:

MANAGER
PRESIDENT
CLERK
ANALYST

Display the job titles of deptno 10 and 20 including duplicates:

```
SELECT job FROM emp WHERE deptno=10  
UNION ALL  
SELECT job FROM emp WHERE deptno=20;
```

Output:

MANAGER
PRESIDENT
CLERK
CLERK
MANAGER
ANALYST

Display common job titles of deptno 10 and 20:

```
SELECT job FROM emp WHERE deptno=10  
INTERSECT  
SELECT job FROM emp WHERE deptno=20;
```

Output:

CLERK
MANAGER

Display specific job titles of deptno 10:

```
SELECT job FROM emp WHERE deptno=10  
MINUS  
SELECT job FROM emp WHERE deptno=20;
```

Output:
PRESIDENT

Display specific job titles of deptno 20:

**SELECT job FROM emp WHERE deptno=20
MINUS
SELECT job FROM emp WHERE deptno=10;**

Output:
ANALYST

Example:

Branch-1:

Customer1

CID	CNAME
1	A
2	B
3	C

Branch-2:

Customer2

CID	CNAME
4	D
2	B
5	E

Display all customers of branch-1 and branch-2 without duplicates:

**SELECT cid, cname FROM customer1
UNION
SELECT cid, cname FROM customer2;**

1	A
2	B
3	C

4	D
5	E

Display all customers of branch-1 and branch-2 including duplicates:

```
SELECT cid, cname FROM customer1
UNION ALL
SELECT cid, cname FROM customer2;
```

1	A
2	B
3	C
4	D
2	B
5	E

Display common customers of branch-1 and branch-2:

```
SELECT cid, cname FROM customer1
INTERSECT
SELECT cid, cname FROM customer2;
```

CID	CNAME
2	B

Display the customers who are visiting branch-1 only:

```
SELECT cid, cname FROM customer1
MINUS
SELECT cid, cname FROM customer2;
```

CID	CNAME
-----	-------

1	A
3	C

Display the customers who are visiting branch-2 only:

```
SELECT cid, cname FROM customer2
MINUS
SELECT cid, cname FROM customer1;
```

CID	CNAME
4	D
5	E

emp_ind

empno	ename
1001	A
1002	B

emp_us

empno	ename
5001	C
5002	D

Display all emps who are working for india and us:

```
SELECT * FROM emp_ind
UNION
SELECT * FROM emp_us;
```

empno	ename
1001	A
1002	B
5001	C
5002	D

Differences b/w UNION and UNION ALL:

UNION	does not give duplicates SLOWER [BECAUSE CHECKING FOR DUPLICATES]
UNION ALL	gives duplicates FASTER

Rules of SET OPERATORS:

- Number of columns in both SELECT QUERIES must be same

SELECT cid FROM customer1

UNION

SELECT cid,cname FROM customer2;

Output:

ERROR

- corresponding columns data types must be same in both select queries.

Example:

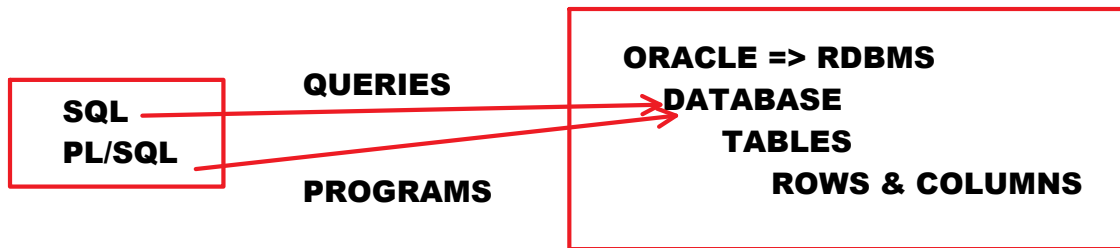
SELECT cid, cname FROM customer1

UNION

SELECT cname, cid FROM customer2;

Output:

ERROR



SQL:

- **Structured Query language**
- **It is a query language.**
- **Non-Procedural Language [no programs]**
- **queries.**

PL/SQL:

- **PL => Procedural Language**
- **it is programming language**
- **Procedural Language.**
- **programs.**

SQL sub languages:

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

Built-In Functions:

String Functions	upper() lower() initcap() Substr() Instr() Lpad() Rpad() Ltrim() Rtrim() Trim() Replace() Translate()
Conversion	to_char() to_date() to_number()
Aggregate / Group	sum() avg() max() min() count()
Number	trunc() round() mod()
Date	sysdate timestamp add_months() months_between()
Analytic	rank() dense_rank() row_number()
Other	NVL() NVL2()

Clauses:

Execution Order [oracle 21c]:

```

FROM emp
WHERE sal>3000
GROUP BY deptno
HAVING sum(sal)>10000
SELECT ename, sal
DISTINCT job
ORDER BY sal DESC
OFFSET 4 ROWS
FETCH NEXT 4 ROWS ONLY

```

Joins:

gola:

used to retrieve the data from multiple tables

types of joins:

```

inner join  => matched records only
  equi      => based on =
  non-equi  => based on other than =
outer join  => matched + unmatched

```

left outer => matched + um from left
right outer => matched + um from right
full outer => matched + um from L & R
self => a table joined to itself
cross => each record in 1 table will be joined with every record in another

Sub Queries:

a query which is another query

Types of sub queries:

Non-Correlated => first inner, inner => 1 time

single row SQ => SQ 1 row

multi row SQ => SQL multiple rows

inline view => FROM

scalar => SELECT

Correlated => first outer => inner => mutilpe times

outer => inner

<=

Constraints:

PK => no dups, no nulls

NOT NULL => no nulls

UNIQUE => no dups

CHECK => to apply our own condn

DEFAULT => to apply default value

REFERENCES [FK] => FK accepts PK values of another table

Set Operators:

UNION

UNION ALL

INTERSECT

MINUS