Thursday, May 19, 2022 9:03 PM

Notes Link:

https://bit.ly/oracle9am

Oracle DataBase 19c installation process

https://bit.ly/oracleinstall

ORACLE

Thursday, May 19, 2022 11:04 AM

ORACLE

SQL
PL/SQL

Data:
is a raw collection of facts about people, places, things ...etc

Ravi
567
25
1234

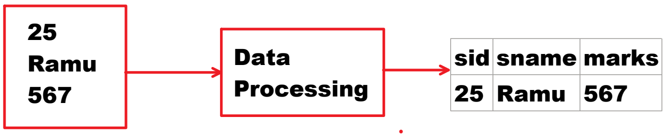
clerk
6000

nareshit
5
3
100
45.6
50.3

smartphone
20000

Data

Information



sid	sname	marks
25	Ramu	567

Data:

- **Data is a raw collection of facts about people, places, things ..etc.**
- **Data is unprocessed one.**
- **It is not in meaningful form.**

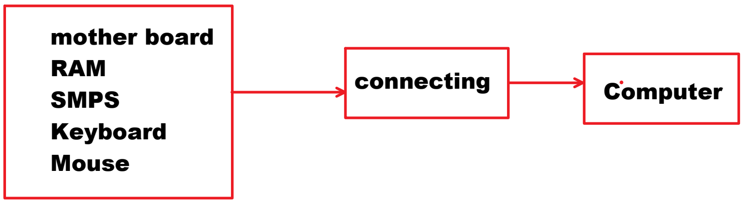
Information:

- **If data is arranged in meaningful form then it is called "Information".**
- **It is processed one.**
- **It is in meaningful form.**

Data can be divided into 2 types:

Structured Data	**if data is in the form of letters, digits & symbols** **Ex: 1234 h.no.1-2-123/A**
• **Unstructured Data**	**if data is in the form of audios, images, videos, documents, animations then it is called**

	unstructured data.



Database:

Bank DB

- Customer
- Transaction
- Product
- Employee

College DB

- Student
- Marks
- Fee
- Library
- Staff

Online Shopping DB

- Product
- Customer
- Order

- **Database => complete details of an organization**

College database

STUDENT

sid	sname	scity
1001	Ramu	Hyd

MARKS

sid	M1	M2	M3
1001	70	80	60

FEE

sid	total_fee	paid_fee	balance
1001	15000	10000	5000

Interrelated data

- **Database is a collection of interrelated data in an organized form.**
- **Database contains records and fields**

Table | Entity

STUDENT

sid	sname	scity
1001	Ramu	Hyd

Column (or) field (or) attribute (or) property

Row (Or) Record (or) Tuple (or) Entity Instance

Table => collection of rows & columns

Record => is a collection of field values.

Field => holds individual values

DBMS:

- **DataBase Management System / Software.**
- **DBMS is a software that is used to create & maintain the database.**
- **It allows us to store, manipulate & retrieve the data of database.**
- **manipulate => add / delete / modify**
- **retrieve => opening existing data**

Before 1960s => store manually in books
1960s => FMS
1970s => Hierarchical & Network DBMS

1976 => E.F.Codd => RDBMS => 12 Rules

1979 => RDBMS => ORACLE

RDBMS:

- **Relation => Table**
- **It is a software that is used to create & maintain the database in the form of tables.**

Ex:
ORACLE DB2 SQL SERVER My SQL Postgre SQL

Metadata:

- **Data about the data.**
- **It describes about the data.**

EX: field name, table name, data type, field size

Data Type
field size
Number(4)

Ex:
STUDENT

sid	sname	scity
25	Ramu	Hyd

Varchar2(20)
20 chars

max 4 digits

9999 ✓

X 10000 5

ORACLE

- **ORACLE is Relational DataBase Management Software [RDBMS].**
- **Relation => Table**
- **It is used to create & maintain the database in the form of tables.**
- **This software allows us to store, manipulate &**

retrieve the data of database.

- **manipulate => add, delete & modify**
- **Ex:**
  - **emp joined in organization => add**
  - **emp got promotion => modify**
  - **emp left from org => delete**
- **2nd version introduced in 1979.**
- **Latest version is: Oracle 21C.**

RDBMS

E.F.Codd => 1976

Larry Ellison => Founder of ORACLE

1977 => Software Development Laboratories
1979 => Relational Software Inc. => ORACLE
1983 => ORACLE carp.

ORACLE
Database

To communicate with ORACLE Database we use 2 Languages:

- **SQL**
- **PL/SQL [Procedural Language]**

SQL:

- **SQL => Structured Query Language.**
- **It is used to write the queries.**
- **Query is a request that is sent to Database Server.**
- **Queries are written to communicate with Database.**
- **SQL is a Non-Procedural Language.**
  It means, we will not write any set of statements.
- **SQL is 4GL [4th Generation Language].**
  In 4GLs, we much focus on what to do rather than how to do.
- **SQL is Unified Language. It is common for many RDBMSs.**

ORACLE	SQL SERVER	DB2
SQL	SQL	SQL

C / JAVA

Programming Languages

Software
=> programs

In C:
Function:
a set of statements

In Java:
Method:
a set of statements

In PL/SQL:

Procedure
a set of statements

Function
a set of statement

- **SQL provides operators to perform operations.**
- **provides built-in functions.**
- **provides JOINS concept to retrieve the data from multiple tables.**

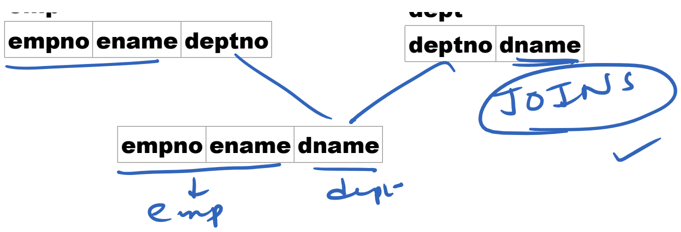
emp

empno	ename	deptno

dept

deptno	dname

JOINS



- provides Sub Queries.

SQL provides 5 sub languages:

DDL **Data Definition Language** **deals with metadata**	**CREATE** **ALTER** **DROP** **TRUNCATE** **RENAME** **FLASHBACK [Oracle 10g]** **PURGE [Oracle 10g]**
DRL / DQL **Data Retrieval Language** **Data Query Language** **deals with data retrievals**	**SELECT**

DML
Data Manipulation Language
 	
deals with data
	INSERT
	UPDATE
	DELETE
	
	INSERT ALL [Oracle 9i]
	MERGE [Oracle 9i]

DCL / ACL **Data Control Language** **Accessing Control Language**	**GRANT** **REVOKE**
TCL **Transaction Control Language**	**COMMIT** **ROLLBACK** **SAVEPOINT**

c##oracle11am
emp

GRANT
REVOKE

c##oracle6pm

DDL Commands:

- **CREATE**
- **ALTER**
- **DROP**
- **TRUNCATE**
- **RENAME**
- **FLASHBACK**
- **PURGE**

CREATE:

- **is used create Database Objects like tables, views, indexes ... etc.**

DB Objects

- **Table**
- **View**
- **Index**
- **Sequence**
- **Synonym**
- **Materialized view**
- **Procedure**
- **Function**
- **Package**
- **Trigger**

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>,
..........................
...........]
);
```

Data Types in SQL:

- **data type tells the type of data which a column a can hold.**

ORACLE SQL provides following data types:

Number related	✓*** **Number(p)** **int** **integer** } → Integer ✓*** **Number(p,s)** **float** **binary_float** **binary_double** } → Floating Point

Character related	**Char(n)**
	Varchar2(n)
	LONG
	CLOB
	
	nChar(n)
	nVarchar2(n)
	nCLOB

Date & time related	**Date** **Timestamp [Oracle 9i]**
Binary related	**BFILE** **BLOB**

23
567
1234

6000.00

67.89

Number Related Data Types:

Number(p):

- **p => precision => max no of digits**
- **used to hold integer values.**
- **precision range: 1 to 38**
- **number(38) => -9999...999 38 digits to 9999....999 38 digits**

Ex:

```
sid          Number(4)    =>     -9999 to 9999
------
1234
1235
1236
9999
10000 => ERROR
```

```
maths_marks   Number(3)  -999 to 999               Max marks: 100

                                                    0 to 100
cid            Number(6)
------
123456                           -999999 to 999999
123457
```

```
Mobile_num  Number(10)
Aadhar_num Number(12)
Credit_card_num Number(16)
```

Number(p,s):

- p => precision => max no of digits
- s => scale   => max no of decimal places
- used to hold floating point values.
- precision => 1 to 38
- scale => -84 to 127

Ex:

```
sal             Number(8,2) => -999999.99 to 999999.99
----------
12000.00
15000.00
11000.00
                                                  100.00
avrg            Number(5,2) => -999.99 to 999.99
----------
56.78
83.24              number(2,1)  =>  -9.9 to 9.9
```

```
height
-----------
5.6
5.4
5.9
6.0
```

Character Related Data types:

Char(n)
Varchar2(n)
LONG
CLOB

nChar(n)
nVarchar2(n)
nCLOB

Char(n):

- used to hold a set of characters [string]
- Fixed Length Data Type.
- Max size: 2000 bytes
- Default size: 1 Ex: gender char =>

Varchar2(n):

- used to hold a set of characters [string].
- Variable length char data type.
- Max size: 4000 bytes
- Default size => No default size
- Ex: ename varchar2 => ERROR

Ex:

Fixed Length

Variable length

TEST5

f1 Char(10)	f2 Varchar2(10)
ramu6spaces	ramu
naresh4spaces	naresh
sai7spaces	sai

10
10
10

4
6
7

PAN_CARD_NUM Char(10)

ABCD512345

Gender Char(1) / Char

M
F

Country_Code Char(3)

IND
AUS
WIN
PAK
USA

State_Code Char(2)

TS
AP
WB
MH

Vehicle_num Char(10)

TS02AA1234

ename Varchar2(10)
----
arun
sravan
sai
naresh

pname Varchar2(10)
----
laptop
hard disk
keyboard
mouse

LONG:

- is used to hold a set of chars (string)
- max size: 2GB
- It has some limitations / restrictions:
  - we cannot use built-in functions on LONG column
  - In one table, one column only we can declare as LONG type.

CLOB:

- Character Large Object.
- It is used to hold a set of chars [string].
- Max size: 4GB

Example:

```
Customer_feedback    CLOB
-----------------------------

Complaints   CLOB

Experience_summary   CLOB
```

Data Types:

Number(p)	precision

```
maths_marks     Number(3)

cust_id             Number(6)
-------------
123456
123457



avrg                    Number(5,2)
--------------
100.00
```

Character Related data types:

Char(n)
Varchar2(n)
LONG
CLOB
	ASCII Code Char related
	Single Byte Char related
	English only

nChar(n)
nVarchar2(n)
nCLOB
	UNI code Char related
	Multi-Byte Char Related
	English + other lang chars

In C:

char => 1 Byte => ASCII
256 chars
[0 to 255]

255	1111 1111

In Java:

char => 2 Bytes => UNI
[0 to 65535]
English + other lang chars

n => national

Date & Time Related Data types:

Date
Timestamp [Oracle 9i]

Date:

- is used to hold date values.
- default date format: DD-MON-RR [23-MAY-22 / 23-JAN-22]
- to_date() function is used to insert the date value.
- It can hold date, month, year, hours, minutes & seconds
- But, it cannot hold fractional seconds.
- Fixed Length Data Type.
- Memory: 7 Bytes
- Default time value: 12:00:00 AM (mid night)

RR	2022	2070
00-49	18 => 2018 22 => 2022	18 => 2118 22 => 2122
50-99	78 => 1978 93 => 1993	78 => 2078 93 => 2093

Ex:

DOB   DOJ   DOR   Del_date   Ord_date   trans_date

Timestamp:

- introduced in Oracle 9i version.
- It is extension of date data type.
- Timestamp can hold date, month, year, hour, minute, seconds & fractional seconds.
- Memory: 11 bytes
- Fixed length data type.

Date	Timestamp
cannot hold fractional seconds	can hold fractional seconds
7 Bytes	11 Bytes
by default it will not display the time value.	displays time value by default.

Binary Related:

used to hold multimedia objects like images, audios, videos, documents ... etc.

2 types:

- BFILE
- BLOB

BFILE:

- Binary File Large Object
- memory: 4GB
- it is pointer to multimedia object.
  It holds path of multimedia object

Database

BFILE

↓

emp

empid	ename	ephoto
1001	ramu	d:\photos\ramu.jpg

D:

photos folder

ramu.jpg

- BFILE data type can be also called as "External LOB data type.
- It is not secured one.

BLOB:

- BLOB => Binary Large Object
- Memory: 4GB
-

BLOB

↓

emp

eid	ename	ephoto
1001	ramu	A1234FD76678A67878 7BC76678DA786

D:

photos folder

ramu.jpg

SQL:

DDL:

- CReate
- Alter
- Drop
- Truncate
- Rename
- Flashback
- Purge

Syntax to CREATE the table:

```
CREATA TABLE <table_name>
(
<field_name> <data_type> [constraint <con_name> <con_type>,
<field_name> <data_type> constraint <con_name> <con_type>,
...................................]
);
```

```
<field_name> <data_type> constraint <con_name> <con_type>,
...................................]
);
```

Syntax of INSERT command:

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

Creating User:

From Oracle 12c version onwards, there are 2 types of users:

- **Common User / Global User**
- **Local User**

Creating Common User:
user name must be prefixed with c##
Ex:

ramu	local user
c##ramu	common user

Syntax to create user:

```
CREATE USER <user_name>
IDENTIFIED BY <password>
DEFAULT TABLESPACE <tablespace_name>
QUOTA <size> ON <tablespace_name>;
```

DataBase Administrator [DBA] can create the users.

Log In as DBA:

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

```
CREATE USER c##batch11am
IDENTIFIED BY nareshit
DEFAULT TABLESPACE users
QUOTA unlimited ON users;


GRANT connect TO c##batch11am;

conn c##batch11am/nareshit
connected

show user
```

```
c##batcha11am

CREATE TABLE t1
(
f1 number(4)
);
ERROR:

conn system/nareshit
connected

GRANT resource TO c##batch11am;
```

Log in as DBA
CReate user
GRANT connect,resource TO c##batch11am;

connect	for login
resource	for creating table

Examples on creating tables:

STUDENT

sid	sname	M1
1001	A	45
1002	B	78
1003	C	66

sid	Number(4)
sname	Varchar2(10)
M1	Number(3)

creating table:

```
CREATE TABLE student
(
sid Number(4),
sname Varchar2(10),
M1 Number(3)
);

Table created.
```

Inserting a record:

```
INSERT INTO student VALUES(1001,'A',45);

INSERT INTO student VALUES(1002,'B',78);
```

Inserting multiple records using parameters:

```
INSERT INTO student
VALUES(&sid,'&sname',&m1);

enter value for sid: 1003
enter value for sname:ramu
enter value for m1:55
```

/	run	it runs recent command

```
/
enter value for sid: 1004
enter value for sname:kiran
enter value for m1:89

/
enter value for sid:
enter value for sname:
enter value for m1:
```

to see table structure:

```
desc student;
```

```
sid     number(4)
sname varchar2(10)
m1       number(3)
```

to see table data:

```
SELECT * FROM student;
```

*	All Columns

to see tables list created by user:

```
Desc User_Tables;


SELECT table_name FROM user_tables;
```

Ex-2:

dd-mon-rr

EMPLOYEE

empno	ename	job	sal	doj
5001	A	manager	9000	25-dec-19
5002	B	clerk	6000	17-aug-21

100000.00

8,2

empno	number(4)
ename	varchar2(12)
job	varchar2(10)
sal	Number(8,2)
doj	date

```
CREATE TABLE employee
(
empno number(4),
ename varchar2(12),
job varchar2(10),
sal number(8,2),
doj date
);
```

Inserting records:

```
INSERT INTO employee
VALUES(5001,'A','manager',9000,'25-dec-2019');

INSERT INTO employee
VALUES(5002,'B',clerk',6000,'17-aug-2021');
```

Inserting multiple records using parameters:

```
INSERT INTO employee
VALUES(&empno,'&ename','&job',&sal,'&doj');
enter value for empno:5003
enter value for ename: C
enter value for job: clerk
enter value for sal: 8000
enter value for doj: 18-jun-2017

/
enter value for empno:
enter value for ename:
enter value for job:
enter value for sal:
enter value for doj:

/
enter value for empno:
enter value for ename:
enter value for job:
enter value for sal:
enter value for doj:
```

Inserting limited column values:

5005	Srinu	manager

```
INSERT INTO employee
VALUES(5005,'Srinu','manager');
ERROR: not enough values

INSERT INTO employee(empno,ename,job)
VALUES(5005,'Srinu','manager');
```

Inserting limited column values by changing the order:

```
INSERT INTO employee(ename,job,empno)
VALUES('Raju','salesman',5006);
```

Constraints:

- Constraint => restrict / limit
- Constraint is a rule that is applied on a column.
- It restricts the user from entering invalid data.

Max Marks:100
0 to 100

Check

m1>=0
AND
m1<=100

student

sid	sname	m1
1001	A	78
1002	B	56
1003	C	496

Invalid

ORACLE SQL provides following constraints:

ORACLE SQL provides following constraints:

- **Primary Key**
- **Unique**
- **Not Null**
- **Check**
- **Default**
- **References [Foreign Key]**

1003	**C**	**496**

→ Invalid

gender

M
M
Fprimary
F
Z → Invalid

Primary Key:

- **does not accept duplicate values.**
- **does not accept null values.**

Ex:

STUDENT
PK

sid	**sname**	**scity**
1001	**Raju**	**Hyd**
1002	**Kiran**	**Hyd**
1003	**Raju**	**Delhi**
	Arun	**Mumbai**
1001	**Sravan**	**Pune**

× null

duplicate ×

PK

Acno	**Name**	**City**
	A	Hyd
1001	A	Hyd
1001	B	Pune

× null

✓

× duplicate

Not Null:

- **It does not accept NULL values.**
- **It accepts duplicate values.**

Ex:

STUDENT

NOT NULL

sid	**sname**	**marks**
1001	**Ramu**	**567**
1002	**Ramu**	**456**
1003		**789**

duplicate ✓

null ×

UNIQUE:

- **It does not accept duplicate values.**
- **It accepts NULL values.**

EX:

STUDENT
UNIQUE

sid	**sname**	**scity**
1234	**Ramu**	**Hyd**
	Arun	**Mumbai**
1234	**Sai**	**HYd**

null ✓

× duplicate

Customer

UNIQUE ↑ null ✓

UNIQUE ↑

Customer

null ✓ UNIQUE UNIQUE

cid	cname	mail_id	mobile_num
1001	RAMU		9012345678
1002	SAI		9012345678

duplicate ✗

Constraint	Null	Duplicate
Primary Key	NO	NO
NOT NULL	NO	YES
UNIQUE	YES	NO

Primary Key = Unique + Not Null

Check:
is used to apply our own conditions on column.

Check (M1>=0 AND M1<=100)

sid	sname	m1
101	A	56
102	B	78
103	C	654

Invalid ERROR

max marks: 100
0 to 100

Check (Gender='M' OR Gender='F')

GENDER

M
F
M
M
F
F
Z

ERROR

Default:

- **used to apply default value to a column.**

Default 'NareshIT' Default 'Hyd' Default 15000

sid	sname	college_name	college_City	FEE
1001	Ramu	NareshIT	Hyd	15000
1002	A	NareshIT	Hyd	15000
1003	B	NareshIT	Hyd	15000
1004	C	NareshIT	Hyd	5000

REFERENCES [Foreign Key]:

Foreign Key refers to primary key values of another table.

another table.

COURSE

PK

cid	cname
10	JAVA
20	PYTHON
30	HTML

STUDENT

Foreign Key

sid	sname	cid
5001	A	30
5002	B	10
5003	C	90

✓ ✓

ERROR

Examples on Constraints:

STUDENT

sid	sname	M1
1	A	45
2	B	56

sid	Number(4)	Primary Key
sname	Varchar2(10)	Not Null
M1	Number(3)	Check => 0 to 100

```
CREATE TABLE student
(
sid number(4) primary key,
sname varchar2(10) not null,
m1 number(3) check(m1>=0 and m1<=100)
);

INSERT INTO student VALUES(1001,'A',56);
INSERT INTO student VALUES(1002,'B',56);

INSERT INTO student VALUES(1001,'C',77);
ERROR: unique constraint violated

INSERT INTO student VALUES(null,'A',56);
ERROR: cannot insert NULL into student.sid

INSERT INTO student VALUES(1005,'EE',786);
ERROR: Check Constraint violated

INSERT INTO student VALUES(1006,'A',56);

INSERT INTO student VALUES(1007,null,89);
ERROR: cannot insert null into student.sname
```

Ex-2:

UNIQUE

STUDENT3

default nareshit

default Hyd

default 15000

sid	sname	cname	ccity	fee
1001	A	nareshit	Hyd	15000
1002	B	nareshit	Hyd	15000
	C	nareshit	Hyd	15000

```
CREATE TABLE student3
(
sid number(4) unique,
sname varchar2(10),
cname varchar2(10) default 'NareshIT',
ccity varchar2(10) default 'Hyd',
Fee number(7,2) default 15000
);
```

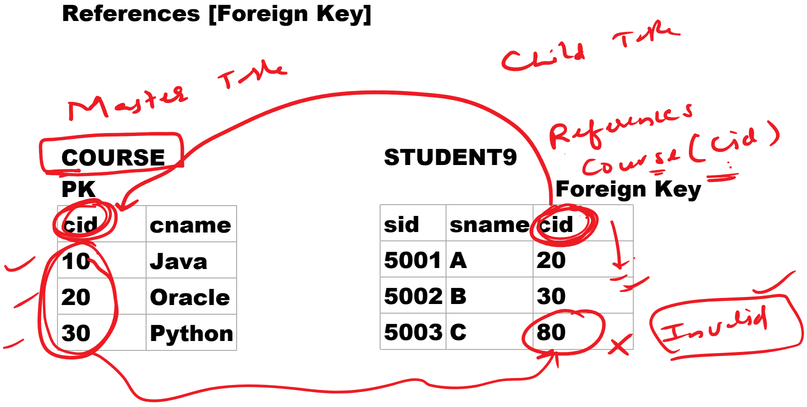
15000.00

7,2

```
INSERT INTO student3 VALUES(1,'A');
ERROR: not enough values

INSERT INTO student3(sid,sname)
VALUES(1,'A');

INSERT INTO student3(sid,sname)
VALUES(1,'B');
ERROR:unique constraint violated

INSERT INTO student3(sid,sname)
VALUES(null,'C');
```

Constraints:
rule => column

Primary Key
Not Null
Unique
Check
Default
References [Foreign Key]



COURSE
PK

cid	cname
10	Java
20	Oracle
30	Python

STUDENT9
Foreign Key

sid	sname	cid
5001	A	20
5002	B	30
5003	C	80

```
CREATE TABLE course
(
cid number(2) primary key,
cname varchar2(10)
);

CREATE TABLE student9
(
sid number(4),
sname varchar2(10),
cid number(2) REFERENCES Course(cid)
);
```

```
INSERT INTO course VALUES(10,'JAVA');

INSERT INTO course VALUES(20,'ORACLE');

INSERT INTO course VALUES(30,'PYTHON');

COMMIT;

INSERT INTO student9 VALUES(1,'A',30);

INSERT INTO student9 VALUES(2,'B',10);

INSERT INTO student9 VALUES(3,'C',80);
ERROR: integrity constraint violated
```

Assignment:

Dept

PK

deptno	dname
10	Accounts
20	Sales
30	HR
40	Research

employee

FK

empno	ename	deptno
1001	A	30
1002	B	40
1003	C	40
1004	D	70

Invalid

```
CREATE TABLE <table_name>
(
<fn> <dt> constraint <con_name> <con_type>,
<fn> <dt> constraint <con_name> <con_type>,
....
);
```

Naming Constraints:

* we can give names to the constraints.

- "constraint" keyword is used to give the constraint name.

STUDENT

PK

sid	sname
1001	A
1002	B
1003	C

disable PK constraint

null

enable PK constraint

Naming Constraints:

* we can give names to the constraints.

- "constraint" keyword is used to give the constraint name.
- When we create table with constraints we have to give constraint names. If we don't give constraint name implicitly ORACLE

gives constraint name by prefixing 6 digit random number with "sys_c"

- **Ex: sys_c543278**

Example:

STUDENT10

sid	sname	M1

field name ↑   DT-type ↑   Con-type ↑   Con-name ↑

sid	number(4)	primary key	c1
sname	varchar2(10)	not null	c2
m1	number(3)	check => 0 to 100	c3

```
CREATE TABLE student10
(
sid number(4) constraint c1 primary key,
sname varchar2(10) constraint c2 not null,
m1 number(3) constraint c3 check(m1>=0 and m1<=100)
);
```

Note:
We cannot give constraint name to DEFAULT constraint

SQL

DDL

- **CREATE**
- **ALTER**
- **DROP**
- **TRUNCATE**
- **RENAME**
- **FLASHBACK**
- **PURGE**

ALTER:

- **ALTER => change**
- **It is used to change structure of the table.**
- **Using ALTER 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
  - ○ add the constraints → ADD CONSTRAINT
  - ○ rename the constraints → RENAME 〃
  - ○ disable the constraints → DISABLE 〃
  - ○ enable the constraints → ENABLE 〃
  - ○ drop the constraints → DROP 〃

STUDENT11

sid	sname	m1

varchar2(20)

Syntax of ALTER command:

Syntax of ALTER command:

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

Example:

STUDENT11

sid	sname

```
CREATE TABLE student11
(
sid number(4),
sname varchar2(10)
);
```

Adding a column [m1 column]:

```
ALTER TABLE student11 ADD m1 number(3);
table altered.
```

Adding multiple columns[m2,m3 columns]:

```
ALTER TABLE student11
ADD(m2 number(3), m3 number(3));
```

```
desc student11;
sid
sname
m1
m2
m3
```

Renaming a Column [m3 to maths]:

```
ALTER TABLE student11
RENAME COLUMN m3 TO maths;
```

Dropping 1 column [maths column]:

```
ALTER TABLE student11 DROP COLUMN maths;
(or)
ALTER TABLE student11 DROP(maths);
```

Dropping multiple columns:

```
ALTER TABLE student11 DROP(m1,m2);
```

Modifying data type [sid number => varchar2]:

```
ALTER TABLE student11
MODIFY sid varchar2(10);
```

Modifying Field size [sname => varchar2(10) => 10 to 20]:

```
ALTER TABLE student11
MODIFY sname varchar2(20);
```

Adding Constraint [Primary key to sid]:

```
ALTER TABLE student11
ADD CONSTRAINT x Primary key(sid);
```

Disabling Constraint [Disable PK]:

```
ALTER TABLE student11
DISABLE CONSTRAINT x;
```

Enabling Constraint [enable PK]:

```
ALTER TABLE student11
ENABLE CONSTRAINT x;
```

Renaming Constraint [x to z]:

```
ALTER TABLE student11
RENAME CONSTRAINT x TO z;
```

Dropping the Constraint [z]:

```
ALTER TABLE student11
DROP CONSTRAINT z;
```

DROP
FLASHBACK [Oracle 10g]
PURGE [Oracle 10g]

Drop:

- **It is used to drop the database objects like tables, views, indexes ..etc.**
- **When we drop the table, it goes to recyclebin.**

Syntax to drop the table:

```
Drop Table <table_name> [Purge];
```

Ex:

```
Drop Table employee;   --10 records
```

To see recyclebin:

```
show recyclebin
    employee
```

Flashback:
used to recollect the dropped table.

Syntax:

```
FLASHBACK TABLE <table_name>
```

```
   TO BEFORE DROP;
```

Ex:

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

Purge:

used to delete the table from recyclebin.

Syntax:

```
   Purge Table <table_name>;
```

Ex:

```
   Purge Table employee;
```

Dropping employee table permanent:

```
Drop TABLE employee;
Purge Table employee;          (or)          Drop Table employee purge;
```

drop	used to drop the tables
flashback	used to restore the table
purge	used to delete from recyclebin

DDL:

CREATE
ALTER
DROP
TRUNCATE
RENAME
FLASHBACK
PURGE

Truncate:

used to delete all records from the table.

Syntax:

```
   Truncate Table <table_name>;
```

Ex:

```
   Truncate table student;
```

Table = structure + data

student

sid	sname
1	A
2	B
3	C

} → Structure
+
} → Data → Truncate

Truncate	deletes table data. structure will not be deleted
Drop	entire table will be deleted

Rename:
used to drop the database objects like tables, views ..etc.

Syntax:

```
Rename <old_name> TO <new_name>;
```

Ex:

```
Rename student11 TO std;
```

DRL / DQL:

- DRL => Data Retrieval Language
- DQL => Data Query Language
- Retrieval => opening existing data
- Query => is a request i.e. sent to Database
- It deals with data retrievals.

ORACLE SQL provides only 1 DRL command:

- SELECT

SELECT:
used to retrieve the data from database.

Syntax:

```
SELECT [DISTINCT] <column_list /*>
FROM <table_list>
[WHERE <condition>]
[GROUP BY <grouping_column_list>]
[HAVING <group_condition>]
[ORDER BY <column_name> Asc/Desc];
```

Execution Order:
FROM
WHERE
GROUP BY
HAVING
SELECT
DISTINCT
ORDER BY

Clause: part of the query

English	SQL
Sentences	queries
Words	clauses

Using SELECT command we can display:

- single record
- a set of records [limited rows]
- all records
- limited columns
- limited rows and columns

- **single record:**

display the emp record whose empno is 7499:

```
SELECT * FROM emp
WHERE empno=7499;
```

- **a set of records [limited rows]**

display all managers records:

```
SELECT * FROM emp
WHERE job='manager';

no rows selected
```

Note:
SQL is not case sensitive language.
String comparison is case sensitive.

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

All Records:

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

Limited Columns:

```
SELLECT ename,job,hiredate FROM emp;
```

limited rows and columns:

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

Operators in SQL:

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

SQL provides following operators:

Arithmetic	**+ - * /**
Relational / Comparison	**< <= > >= = != / <> / ^=**
Logical	**AND OR NOT**
Special	**IN NOT IN BETWEEN AND NOT BETWEEN AND IS NULL IS NOT NULL LIKE NOT LIKE**
SET	**UNION UNION ALL INTERSECT MINUS**
Miscellaneous	**|| => concatenation operator Any All Exists Pivot Unpivot**

In c/java:
5%2 = 1
int/int =int
5/2 = 2

in SQL:
mod(5,2) = 1
5/2 = 2.5

Arithmetic Operators:
are used to perform arithmetic operations
+ - * /

Calculate 100+200:

```
SELECT 100+200 FROM dual;

100+200
---------------
300
```

Dual:

- **it is a predefined table included in "sys" schema [user].**
- **it has one column and 1 row.**
- **Because of it is having one row, always returns 1 value.**
- **used to work with non-database values.**

```
SELECT 10+20+30 FROM dual;

10+20+30
---------------
60
```

```
SELECT 10+20+30 as total FROM dual;

total
-----------
60
```

column alias:

- **Alias means, another name / alternative name.**
- **"as" keyword can be used to give column alias.**
  Using "as" keyword is optional.
- **If we want to maintain the case or to give alias name in multiple words specify alias name in double quotes.**
- **Column alias scope is limited to that query only.**
  It cannot be used in other queries.

```
SELECT 10+20 FROM dual;

10+20
-----------
30

SELECT 10+20 as total FROM dual;

TOTAL
-----------
30

SELECT 10+20 total FROM dual;

TOTAL
-----------
30
```

```
SELECT 10+20 as "total" FROM dual;

total
---------
30
```

```
SELECT 10+20 as total value FROM dual;
ERROR:
```

```
SELECT 10+20 as "total value" FROM dual;

total value
--------------------
30
```

Calculate Annual Salary:

```
SELECT empno,ename,sal,
sal*12 as "Annual Salary"
FROM emp;
```

STUDENT

sid	sname	M1	M2	M3
1001	A	70	60	80
1002	B	50	30	70

```
CREATE TABLE student
(
sid number(4),
sname varchar2(10),
m1 number(3),
m2 number(3),
m3 number(3)
);

INSERT INTO student VALUES(1001,'A',70,60,80);
INSERT INTO student VALUES(1002,'B',50,30,70);
```

Calculate TA, HRA, TAX & GROSS salaries:
TA => 10% on sal
HRA => 20% on sal
TAX => 5% on sal
GROSS => sal+TA+HRA-TAX

```
SELECT empno,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_SAL
FROM emp;
```

Relational Operators / Comparison Operators:

are used to compare 2 values.

<   <=   >   >=   =   != / <> / ^=

Display all managers records:

```
SELECT * FROM emp WHERE job='manager';
no rows selected
```

NOTE:
string comparison is case sensitive. In table job values are in upper case.

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

Display the emp records whose salary is less than 1200:

```
SELECT * FROM emp
WHERE sal<1200;
```

Display the emp records whose salaries are greater than 2500:

```
SELECT * FROM emp
WHERE sal>2500;
```

Display the emp records whose is earning 3000 or more than 3000:

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

Display the emp records who joined after 1981:

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

Display the emp records who joined before 1981:

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

Display all emps records except managers:

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

Logical Operators:

- **are used to perform logical operations.**

AND   OR   NOT

AND	**used to perform Logical AND operations**
OR	**used to perform Logical OR operations**
NOT	**used to perform Logical NOT operations**

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

AND	all conditions should be satisfied
OR	at least one condition should be satisfied

STUDENT

sid	sname	M1	M2	M3

Find result of the student.
Max marks: 100
Min Marks: 40   in each sub for pass

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 all managers and clerks records:

```
SELECT * FROM emp
WHERE job='MANAGER' OR job='CLERK';
```

Display all managers records who are earning more than 2500:

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

Display the emp records who joined in 1981:

hiredate<'1-jan-1981'	joined before 1981
hiredate>'31-dec-1981'	joined after 1981
hiredate>='1-jan-1981' AND hiredate<='31-dec-1981'	joined in 1981

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

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

```
SELECT ename,deptno FROM emp
```

```
WHERE deptno=10 OR deptno=30;
```

Display all emp records except managers:

```
SELECT * FROM emp
WHERE job<>'MANAGER';
(or)
SELECT * FROM emp
WHERE Not(job='MANAGER');
```

c1	Not(C1)
T	F
F	T

Display the emp records whose names are
ALLEN, SCOTT & WARD:

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

Special Operators:
Special Operators are also comparison operators.

```
IN       NOT IN
BETWEEN AND     NOT BETWEEN AND
LIKE         NOT LIKE
IS NULL    IS NOT NULL
```

IN:

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

Syntax:

```
<column_name> IN(<value_list>)
```

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

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

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

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

Display all managers, clerks & analysts records:

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

Display all emps records except managers and clerks:

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

BETWEEN AND:

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

Syntax:

```
<column_name> BETWEEN <lower_value> AND <upper_value>
```

Examples:

Display the emp records whose salary is b/w 1600 and 3000:

```
SELECT * FROM emp
WHERE sal>=1600 AND sal<=3000;
```

(or)

```
SELECT * FROM emp
WHERE sal BETWEEN 1600 AND 3000;
```

Display the emp records who joined in 1981:

```
SELECT ename,hiredate FROM emp
WHERE hiredate  BETWEEN '1-jan-1981' AND '31-dec-1981';
```

Display the emp records who joined in 1980,1981,1982:

```
SELECT ename,hiredate FROM emp
WHERE hiredate BETWEEN '1-jan-1980' AND '31-dec-1982';
```

Display the emp records whose salary is less than 1600 or greater than 3000:

```
SELECT * FROM emp
WHERE sal<1600 OR sal>3000;
```

(or)

```
SELECT * FROM emp
WHERE sal NOT BETWEEN 1600 AND 3000;
```

to search for all jpg files	*.jpg
to search for jpg files which names are started with s	s*.jpg
to search for jpg files which names are having 2nd char as a	?a*.jpg

arun.jpg
kiran.jpg
rama.jpg
charan

wildcard characters

*	0 or any no of chars
?	replaces 1 char

LIKE:

- used to compare column value with text pattern

Syntax:

```
<column_name> LIKE <text_pattern>
```

In SQL we can use 2 wildcard characters:

_	replaces 1 character
%	replaces o or any no of chars

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

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

Display the e,p records whose naes are ended with 'S':

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

Display the emp records whose names are having A char:

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

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

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

Display the emp records whose names 2nd char must be A:

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

Display the emp records whose names 3rd char is M:

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

Display the emp records whose names are having 4 chars:

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

Display the emp records who are getting 3 digit salary:

```
SELECT * FROM emp
WHERE sal LIKE '___';
```

Display the emp records who joined in first 9 days in the month:

```
SELECT * FROM emp
WHERE hiredate LIKE '0%';
```

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

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

Display the emp names which are having _ :

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

Display the emp records which are having %:

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

Null:

- **Null => empty / blank**
- **When the value is unknown we insert NULL value.**
- **Null value can be inserted in 2 styles:**
  - **using NULL keyword**
  - **by inserting limited column values**

Example:

Customer

cid	cname	ccity
5001	Vijay	Hyd
5002	Amar	Mumbai
5003	Ramu	

```
CREATE TABLE customer
(
cid number(4),
cname varchar2(10),
ccity varchar2(10)
);
```

```
INSERT INTO customer VALUES(5001,'Ramu','Hyd');
```

1st way: using NULL keyword:

```
INSERT INTO customer VALUES(5002,'Vijay',null);
```

2nd way: by inserting limited column values:

```
INSERT INTO customer VALUES(5003,'Srinu');
ERROR: not enough values
```

```
INSERT INTO customer(cid,cname)
VALUES(5003,'Srinu');
```

- **If NULL is participated in operation then it returns NULL only.**

  ```
  Ex:   20+null  =  null
        10+50+90+null = null
         20-null  = null
  ```

- **For null comparison we cannot use = (equals) operator. For null comparison use "IS NULL".**

where comm=300	displays whose comm is 300
where comm=null	displays no rows selected
where comm IS null	display whose comm is null

IS NULL:

- **used to compare column value with null value.**

  Syntax:

  ```
  <column_name> IS null
  ```

Display the emp records who are not getting commission:

```
SELECT ename,sal,comm FROM emp
WHERE comm=null;
no rows selected
```

Note: for null comparison = cannot be used

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

Display the emp records who are getting commission:

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

Concatenation Operator:

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

```
SELECT 'raj' || 'kumar' FROM dual;
rajkumar
```

```
SELECT 'raj' || ' ' || 'kumar' FROM dual;
raj kumar
```

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

```
SELECT ename || '   ' || sal as ename_sal FROM emp;
```

SQL

DDL => C A D T R F P
DRL/DQL => SELECT

DML
TCL

DCL

DML:

- Data Manipulation Language
- Manipulation => INSERT / UPDATE / DELETE
- It deals with the data.
- All DML commands are not auto-committed.
  All DDL commands are auto-committed.

TCL:

- Transaction Control Language.
- Transaction => a series of commands

Ex: Withdraw   deposit   check balance   placing order

Withdraw Transaction →

Reads card info
Enter PIN → SELECT
Menu
Withdraw
enter amount: 10000 - SELECT
update the balace update.

Bank DB

acno	name	Pin	balance
1001	A		~~50000~~ 40000

placing order

order
INSERT

customer
Insert

Products
iphone => QIH
5
update 3

Transaction must be successfully completed or aborted [cancelled].

DML:

- Data Manipulation Language
- Manipulation => INSERT / UPDATE / DELETE
- It deals with the data.
- All DML commands are not auto-committed.
  All DDL commands are auto-committed.
- If Transaction is successfully completed, use COMMIT.
- If Transaction is not successfully completed, use ROLLBACK.

TCL:

- Transaction control language
- It deals with the transactions

COMMIT
ROLLBACK
SAVEPOINT

COMMIT [SAVE]:

- used to save the transaction.
- When COMMIT command is executed, all changes of Oracle Instance [RAM] will be applied to oracle database[hard disk]

ROLLBACK [UNDO ALL]:

- used to cancel previous actions.
- It cancels uncommitted actions.
- We cannot use ROLLBACK after COMMIT.

```
INSERT INTO customer VALUES(2001,'AA','HYD');
INSERT INTO customer VALUES(2002,'BB','DELHI');
COMMIT;

INSERT INTO customer VALUES(2003,'CC','HYD');
INSERT INTO customer VALUES(2004,'DD','DELHI');
Rollback;
```

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

DDL command = DDL command + Commit

CREATE t1 => Create + Commit
INSERT
INSERT
INSERT
INSERT
Rollback

CREATE t2
INSERT
INSERT
CREATE t3 => commit
INSERT
INSERT

INSERT
INSERT
Rollback

4 actions will be cancelled

INSERT
CREATE t3 => commit
INSERT
INSERT
Rollback

2 actions will be cancelled

DML commands:

ORACLE SQL provides following DML commands:

- **INSERT**
- **UPDATE**
- **DELETE**
- **INSERT ALL**
- **MERGE**

Update:

- **used to update(modify) the data.**
- **Using this command we can update:**
  - ○ **single value of single record**
  - ○ **multiple values of single record**
  - ○ **a set of records**
  - ○ **all records**
  - ○ **using parameters**

Syntax of UPDATE command:

```
UPDATE <table_name>
SET <col_name>=<value>
[,<col_name>=<value>, <col_name>=<value>,
..
[WHERE <condition>];
```

○ **single value of single record:**

increase 2000 rupees salary to the emp whose empno is 7499:

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

○ **multiple values of single record:**

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

a set of records:

increase 1000 rs to all clerks:

UPDATE emp SET sal=sal+1000
WHERE job='CLERK';

all records:

increase 1000 rs sal to all emps:

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

Increase salary of emps as following:
7499 => increase 1000
7521 => increase 2000
7900 => increase 1500

```
Update emp SET sal=sal+&amount WHERE
empno=&empno;
Enter value for amount: 1000
Enter value for empno: 7499


SQL> /
Enter value for amount: 2000
Enter value for empno: 7521



SQL> /
Enter value for amount: 1500
Enter value for empno: 7900
```

Using UPDATE command we can perform calculations & store the result in table.

STUDENT1

sid	sname	m1	m2	m3	total	avrg
1001	A	50	60	40		
1002	B	77	55	64		

100.00

```
CREATE TABLE student1
(
sid number(4), sname varchar2(10),
m1 number(3), m2 number(3), m3 number(3),
total number(3), avrg number(5,2)
);

INSERT INTO student1(sid,sname,m1,m2,m3)
VALUES(1001,'A',60,50,90);

INSERT INTO student1(sid,sname,m1,m2,m3)
VALUES(1002,'B',66,55,99);

COMMIT;
```

Calculate total and avrg:

```
UPDATE student1 SET total=m1+m2+m3,
avrg=(m1+m2+m3)/3;
```

employee

empno	ename	job	sal	TA	HRA	TAX	GROSS
1001	A	clerk	6000				
1002	B	manager	9000				

Caculate TA, HRA, TAX & Gross:
10% on sal as TA
20% on sal as HRA
5% on sal as TAX
GROSS = sal+TA+HRA-TAX

Increase 20% sal to the emps who are having more than 40 years experience:

```
UPDATE emp SET sal=sal+sal*0.2
WHERE (sysdate-hiredate)/365>40;
```

Transfer all deptno 10 emps to deptno 20:

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

Increase 10% sal, 15% comm to the emps who are getting commission:

```
UPDATE emp SET sal=sal+sal*0.1,
comm=comm+comm*0.15
WHERE comm is not null;
```

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

```
UPDATE emp SET sal=sal+sal*0.2
WHERE hiredate LIKE '%82';
(or)
UPDATE emp SET sal=sal+sal*0.2
WHERE hiredate BETWEEN '1-jan-1982' AND
'31-dec-1982';
```

DELETE:

- **used to delete the records from table.**
- **Using this command we can delete:**
  - **single record**
  - **a set of records**
  - **all records**
  - **using parameters**

Syntax:

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

- **single record:**

delete an emp record whose wmpno is 7499:

```
DELETE FROM emp
WHERE empno=7499;
```

a set of records:

delete all managers records:

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

delete all records:

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

SQL

DDL	CREATE ALTER TRUNCATE RENAME FLASHBACK PURGE DROP
DRL / DQL	SELECT

DML	INSERT
	UPDATE
	DELETE
	INSERT ALL
	MERGE

TCL	ROLLBACK [undo all] COMMIT [save] SAVEPOINT
DCL	GRANT REVOKE

SAVEPOINT:
is used to set margin for rollback.

Syntax:

```
SAVEPOINT <savepoin_name>;
```

Ex:

```
CREATE TABLE t1
INSERT
INSERT
INSERT
INSERT
INSERT
INSERT
Rollback;
```

```
CREATE TABLE t1
SAVEPOINT aaa;
INSERT
INSERT
SAVEPOINT bbb;
INSERT
INSERT
SAVEPOINT ccc;
INSERT
INSERT

Rollback to ccc;  --2 actions cancelled
Rollback to bbb;  --4 actions cancelled
```

Rollback to aaa;

DCL:
- Data Control Language.
- It deals with data accessibility.
- It can be used to implement the security.

ORACLE SQL provides 2 DCL commands:
- GRANT
- REVOKE

c##batch11am
emp

c##batch6pm

GRANT:
- used to grant the permissions on database objects [table,view,...] to other users

Syntax:

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

REVOKE:
- used to cancel the permissions on db objects from users.

Syntax:

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

Example on GRANT & Revoke:

Create 2 users:

Log In as DBA:
username: system
password: nareshit

```
CREATE USER c##userA
IDENTIFIED BY usera
DEFAULT TABLESPACE users
QUOTA unlimited ON users;

CREATE USER c##userB
```

```
IDENTIFIED BY userb
DEFAULT TABLESPACE users
QUOTA unlimited ON users;

GRANT connect,resource TO c##userA, c##userB;
```

c##userA	c##userB

Create table t1
(
f1 number(4),
f2 varchar2(10);
);
 	
insert into t1 values(1,'A');
insert into t1 values(2,'B');
COMMIT;	

	SELECT * FROM c##userA.t1; ERROR: table does not exist
GRANT select ON t1 TO c##userB;	
	SELECT * FROM c##userA.t1; f1 f2 -- -- 1 A 2 B INSERT INTO c##userA.t1 VALUES(3,'C'); ERROR: insufficient privileges DELETE FROM c##userA.t1; ERROR: insufficient privileges UPDATE c##userA.t1 SET f2='Ramu' WHERE f1=1; ERROR: insufficient privileges
GRANT insert,update ON t1 TO c##userB;	

	INSERT INTO c##userA.t1
	VALUES(3,'C');
	1 row created
	
	DELETE FROM c##userA.t1;
	ERROR: insufficient privileges
	
	UPDATE c##userA.t1 SET
	f2='Ramu' WHERE f1=1;

GRANT all ON t1 TO c##userB;	
	DELETE FROM c##userA.t1

ORACLE DB

c##userA
emp

c##userB
emp

c##userC

c##userA	c##userB	c##batch11am
t1		

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

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

-- With Above query c##userA is allowing c##userB to gran permissions to othe rusers

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

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

```
GRANT select ON c##userA.t1
TO c##batch11am;
```

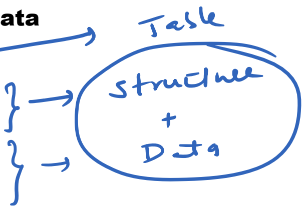
```
SELECT * FROM c##userA.t1;
ERROR: table does not exist
```

Copying Table & Copying Records:

Table => Structure + Data

STUDENT

sid	sname	scity
1001	A	Hyd
1002	B	Mumbai



Copying Table:

Syntax:

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

Create a new table using emp table:

emp

empno	ename	job	sal	hiredate	deptno
1001					
..					
1010					

employee

empno	ename	job	sal

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

Create a new table with managers records:

employee1

empno	ename	job	sal
		MANAGER	
		MANAGER	

```
CREATE TABLE employee1
AS
SELECT empno,ename,job,sal
FROM emp
WHERE job='MANAGER';
```

Copying structure from emp table. Don't copy the data:

```
CREATE TABLE employee3
AS
SELECT empno,ename,job,sal
FROM emp
WHERE 1=2;
```

Copying records:

```
CREATE TABLE customer
(
cid number(4),
cname varchar2(10)
);
```

emp

empno	ename	job	sal
1001			
..			
..			
1010			

customer

cid	cname
1001	—

1002
:
1010 —
copy
=> ✓

Syntax for Copying records:

```
INSERT INTO <table_name>[(<column_list>)]
<SELECT query>;
```

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

INSERT ALL:

- used to insert multiple records in multiple tables or single table.
- Using INSERT command we can insert one record at a time & in one table only. But, using INSERT ALL, we can insert multiple records into multiple tables or single table.

- It avoids of writing multiple INSERT commands.
- INSERT ALL can be used in 2 styles:
  - 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)
into <table_name>[(<column_list>)] values(value_list)
.
.
<SELECT query>;
```

emp

empno	ename	job	sal	hiredate	deptno
1001					
..					
1010					

emp1

empno	enamejob	sal

emp2

empno	enamejob	sal

emp3

empno	enamejob	sal

Create emp1,emp2,emp3 with empno,ename,job,sal columns from emp & without data:

```
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;


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;
```

=

```
INSERT INTO emp1 VALUES
SELECT empno,ename,job,sal
FROM emp;

INSERT INTO emp2 VALUES
SELECT empno,ename,job,sal
FROM emp;

INSERT INTO emp3 VALUES
SELECT empno,ename,job,sal
FROM emp;
```

48 rows created

16*3 = 48

16*3 = 48

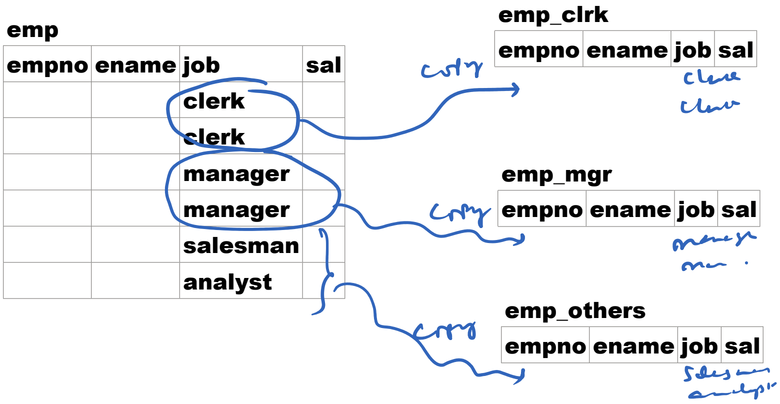
```
SELECT empno,ename,job,sal
FROM emp;
```

if else if

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>;
```



emp

empno	ename	job	sal
		clerk	
		clerk	
		manager	
		manager	
		salesman	
		analyst	

emp_clrk

empno	ename	job	sal

emp_mgr

empno	ename	job	sal

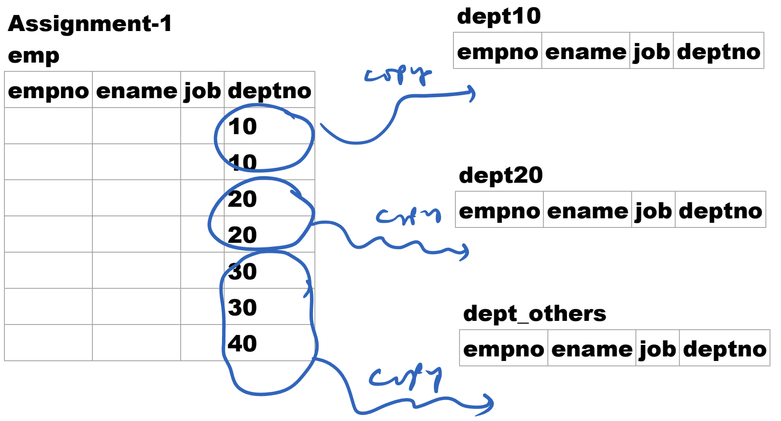
emp_others

empno	ename	job	sal

CReate 3 tables with the names
emp_mgr
emp_clrk
emp_others
with 4 columns empno,ename,job,sal from emp table
without data:

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

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

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

```
WHERE 1=2;


INSERT ALL
WHEN job='MANAGER' THEN
into emp_mgr values(empno,ename,job,sal)
WHEN job='CLERK' THEN
into emp_clrk values(empno,ename,job,sal)
ELSE
into emp_others values(empno,ename,job,sal)
SELECT empno,ename,job,sal FROM emp;
```



Assignment-1

emp

empno	ename	job	deptno
			10
			10
			20
			20
			30
			30
			40

dept10

empno	ename	job	deptno

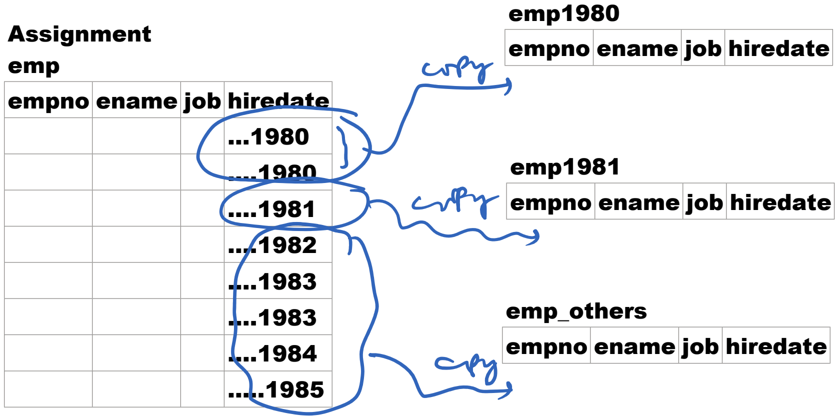
dept20

empno	ename	job	deptno

dept_others

empno	ename	job	deptno

```
WHEN hiredate BETWEEN '1-jan-1980' AND '31-dec-1980' THEN
  into emp1980
```



Assignment

emp

empno	ename	job	hiredate
			...1980
			1980
			1981
			1982
			1983
			1983
			1984
			1985

emp1980

empno	ename	job	hiredate

emp1981

empno	ename	job	hiredate

emp_others

empno	ename	job	hiredate

Replication:
The process of making duplicate copies is called "Replication".

Replica => Duplicate Copy

Types of Databases:

2 types:

- OLTP
- OLAP

OLTP	OLAP /DWH / DSS
• OnLine Transaction Processing	• OnLine Analytical Processing
• used to perform day-to-day operations	• used for data analysis. It maintains historical data.

• In this we perform CRUD operations
C => CREATE
R => READ
U => UPDATE
D => DELETE
	• used to perform READ operations only.

SBI Bank

2018	2017
2019	2018
2020	2019
2021	2020
	2021
2022-23	2022

OLTP

Customer1

cid	cname	ccity
1	A	Bangalore
2	B	Mumbai
3	C	Delhi
4	D	Bangalore
5	E	Pune

matched

update

not matched

INSERT

OLAP

Customer2

cid	cname	ccity
1	A	Hyd
2	B	Mumbai
3	C	Delhi

Merge:

- is used to apply changes of one table to its replica
- Merge = UPDATE + INSERT
- Merge is a combination of update & insert commands.
- It can be also called as "UPSERT" command.

S.cid = t.cid

OLTP

Customer1 S

cid	cname	ccity
1	A	~~Hyd~~ Bangalore
2	B	~~Mumbai~~ Kolkata
3	C	Delhi
4	D	Bangalore
5	E	Pune

matched

update

not matched

Insert

OLAP

Customer2 t

cid	cname	ccity
1	A	Hyd
2	B	Mumbai
3	C	Delhi

Syntax of MERGE command:

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

```
<UPDATE query>
WHEN not matched THEN
<INSERT query>;
```

```
CREATE TABLE customer1
(
cid number(4),
cname varchar2(10),
ccity varchar2(10)
);

INSERT INTO customer1 VALUES(1,'A','Hyd');
INSERT INTO customer1 VALUES(2,'B','Mumbai');
INSERT INTO customer1 VALUES(3,'C','Delhi');
commit;


CREATE TABLE customer2
AS
SELECT * FROM customer1;


INSERT INTO customer1 VALUES(4,'D','Bangalore');
INSERT INTO customer1 VALUES(5,'E','Pune');

UPDATE customer1 SET ccity='Bangalore'
WHERE cid=1;
UPDATE customer1 SET ccity='Kolkata'
WHERE cid=2;
COMMIT;


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);
```

Built-In Functions

Monday, June 6, 2022 12:09 PM

Built-In Functions:

- Function => Task / Action
- ORACLE developers defined some functions & placed in ORACLE database. These functions are called "Built-In Functions / Predefined Functions."

ORACLE SQL provides following built-in functions:

- String Functions / text Functions
- Conversion Functions
- Aggregate Functions / group functions
- Math Functions / Number Functions
- Date Functions
- Miscellaneous Functions

- **String Functions / text Functions:**

lower()
upper()
initcap()
length()
concat()

Substr()
Instr()

Ltrim()
Rtrim()
Trim()

Lpad()
Rpad()

Reverse()
Replace()
Translate()

Soundex()
ASCII()
Chr()

lower():
used to convert the string to lower case.

Syntax:

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

Ex:

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

Upper():
used to convert the string to upper case.

Syntax:
Upper(<string>)

Ex:

Upper('ramu')	RAMU
Upper('ravi teja')	RAVI TEJA

Initcap():
used to get every word starting letter as capital.

Syntax:
Initcap(<string>)

Ex:

Initcap('RAMU')	Ramu
Initcap('RAVI KUMAR')	Ravi Kumar

length():
used to find length of the string.

Syntax:
length(<string>)

Ex:

length('sai')	3
length('naresh')	6

concat():
used to concatenate (Combine) 2 strings.

EX:

concat('raj','kumar')	rajkumar
concat(concat('raj',' '),'kumar') (or)	raj kumar

'raj' || ' ' || 'kumar'	

Convert emp names to initcap case:

UPDATE emp SET ename=initcap(ename);

Display the emp records whose names are having 4 chars:

```
SELECT * FROM emp
WHERE ename LIKE '____';
(or)
SELECT * FROM emp
WHERE length(ename)=4;
```

Display the emp names which are having more than 4 chars:

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

Substr():
is used to get sub string from the string.

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

Exs:

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

position value can be given as -ve

Substr('Raj Kumar',5)	Kumar
SubStr('Raj Kumar',6)	umar

+ve	from left side pos num

Substr('Raj Kumar',5)	Kumar
SubStr('Raj Kumar',6)	umar
SubStr('Raj Kumar',1,3)	Raj
SubStr('Raj Kumar',1,5)	Raj K
SubStr('Raj Kumar',6,3)	uma
SubStr('Raj Kumar',-5)	Kumar
SubStr('Raj Kumar',-4,3)	uma
SubStr('Raj Kumar',-4)	umar

+ve	from left side pos num
-ve	from right side pos num

Instr():
used to get position of sub string.

Syntax:
Instr(<string>,<substring>[,<position>,<occurrence>])

position	1
occurrence	1

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
t	h	i	s		i	s		h	i	s		w	i	s	h
-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Instr('sai teja','teja')	returns 5
Instr('sai teja','naresh')	returns 0
Instr('this is his wish','is')	3
Instr('this is his wish','is',1,2)	6
Instr('this is his wish','is',4,2)	10
Instr('this is his wish','is',-1)	14
Instr('this is his wish','is',-1,2)	10
Instr('this is his wish','is',-4)	10
Instr('this is his wish','is',-4,2)	6

Generate email ids for all emps by taking empname first 3 chars & empno last 3 chars as username:

```
ALTER TABLE emp add mail_id varchar2(50);

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

Display the emp names in which starting letter and ending letter as same:

```
INSERT INTO emp(empno,ename) VALUES(4001,'DAVID');
INSERT INTO emp(empno,ename) VALUES(4002,'SYMONDS');
```

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

Display the emp records whose name is started with S:

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

Display the emp records whose name is ended with RD:

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

Display the emp records whose names are having R char:

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

Lpad() & Rpad():

- pad => fill

Lpad():

- used to fill specified char/chars from left side.

Syntax:

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

3rd arg default	space

Rpad():

- used to fill specified char/chars from right side.

Syntax:

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

3rd arg default	space

Exs:

Lpad('raju',8,'*')	****raju
Lpad('sai',10,'@')	@@@@@@@sai
Rpad('sai',10,'@')	sai@@@@@@@
Lpad('sai',10,'#$')	#$#$#$#sai
Lpad('Ravi,10)	6spacesRavi
Rpad('Sai',8)	Sai5spaces
Lpad('A',10,'A')	AAAAAAAAAA

1234567890

amount debited from XXXXXX7890

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

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

Ltrim():

- trim => remove
- used to remove unwanted chars from left side.

Syntax:

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

Rtrim():

- used to remove unwanted chars from right side.

Syntax:

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

2nd arg default	space

Exs:

Ltrim('@@@raju@@@','@')	raju@@@
Rtrim('@@@raju@@@','@')	@@@raju
Ltrim(' sai ')	sai3spaces
Rtrim(' sai ')	3spacessai

Trim():

we can remove left side or right side or both sides unwanted chars

Syntax:

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

Trim(Leading '@' FROM '@@@raju@@@')	raju@@@
Trim(Trailing '@' FROM '@@@raju@@@')	@@@raju
Trim(Both '@' FROM '@@@raju@@@')	raju
Trim(' ravi ')	ravi
Trim('@' FROM '@@@raju@@@')	raju

Aggregate Functions [group Functions]:

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

sum():
used to find sum of values of a column

Syntax:

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

Ex:
find sum of salaries of all emps:

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

find sum of salaries of deptno 10:

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

find sum of salaries of all managers:

```
SELECT sum(sal) from emp
where job='MANAGER';
```

Avg():
used to find average value

Syntax:

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

Ex:

Find avg salary of all emps:

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

Find avg salary of deptno 10:

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

max():
is used to find max value

Syntax:
max(<column>)

Ex:
find max salary in all emps:

```
select max(Sal) from emp;
```

find max sal in deptno 20:

```
select max(Sal) from emp where
deptno=20;
```

min():
used to find min value

syntax:
min(<column>)

Ex:
find min salary in all emps:

```
select min(sal) from emp;
```

count():

used to count no of column values or no of records.

Syntax:

```
   count(<column_name>)
```

find how many emps are getting commission:

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

find no of records in emp table:

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

find no of emps in deptno 10:

```
SELECT count(*) FROM emp
WHERE deptno=10;
```

Math / Number Functions:

power()
sqrt()
sin()
cos()
tan()

sign()
abs()

mod()
ceil()
floor()
trunc()
round()

power():
used to find power values

syntax:

power(number,power)

Ex:

power(2,3)	**8**

sqrt():
used to find square root value

syntax:
sqrt(number)

ex:

sqrt(100)	**10**

sin():
used to find sine values

syntax:
sin(angle)

cos():
used to find cosine values

syntax:
cos(angle)

tan()
used to find tangent values

syntax:
tan(angle)

sin 90	**sin(90*3.14/180)**
cos 0	**cos(0*3.14/180)**

sign():

used to check whether num is +ve or -ve or zero

syntax:
sign(number)

ex:

sign(5)	**1**
sign(-5)	**-1**
sign(0)	**0**

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

syntax:
abs(number)

Ex:

abs(5)	**5**
abs(-5)	**5**

mod():
used to get remainder values

syntax:
mod(number,divisor)

Ex:

mod(5,2)	**1**
mod(20,7)	**6**

floor():
used to get lower integer value

syntax:
floor(number)

ceil():
used to get upper integer value

syntax:
ceil(number)

floor(123.4567)	**123 & 124 123**
ceil(123.4567)	**124**

Trunc():
used to remove decimal places

syntax:
trunc(number,no_of_decimal_places)

trunc always returns lower value

ex:

trunc(123.45678)	**123**
trunc(123.45678,2)	**123.45**
trunc(123.45678,3)	**123.456**
trunc(123.4567,-1)	**120 & 130 120**
trunc(1234.567,-2)	**1200 & 1300 1200**
trunc(1234.567,-3)	**1000 & 2000 1000**

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

10,20,30,...,100,110,120,130

round():
used to get rounded values.

value is avrg or above avrg	**upper**

value is below avrg	lower

percentage 56.5

```
---------------------
56.78 → 57
78.39 → 78
66.52 → 67
77.64 → 78
```

round(123.45678)	123
round(123.78234)	124
round(123.45678,2)	123.46
round(123.45378,2)	123.45
round(123.45678,-1)	120 & 130 avrg => 125 120
round(127.45678,-1)	120 & 130 avrg => 125 130

round(123.45678,-1)	120 & 130 avrg => 125 120
round(127.45678,-1)	120 & 130 avrg => 125 130
trunc(123.45678,-1)	120 & 130 avrg => 125 120
trunc(127.45678,-1)	120 & 130 avrg => 125 120

Date Functions:

sysdate
systimestamp
add_months()
last_day()
next_day()
months_between()

sysdate:
used to get current system date

systimestamp:
used to get current system date & time

```
SELECT sysdate FROM dual;
8-jun-22

select systimestamp from dual;
8-jun-22 11:57:15.678900 AM
```

Add_Months():
used to add/subtract months to/from a date.

syntax:
Add_Months(date,no_of_months)

Add 2 days to today's date:
8-jun-22

```
SELECT sysdate+2 FROM dual;
10-jun-22
```

Add 2 months to today's date:

```
SELECT add_months(sysdate,2) FROM dual;
8-aug-22
```

dob

25-dec-2000

dor

SELECT add_months(sysdate,2) FROM dual;
8-aug-22

dor

Add 2 years to today's date:
SELECT add_months(sysdate,2*12) FROM dual;
8-jun-2024

Subtract 2 days from today's date:
SELECT sysdate-2 FROM dual;
6-jun-22

Subtract 2 months from today's date:
SELECT add_months(sysdate,-2) from dual;

Subtract 2 years from today's date:
SELECT add_months(sysdate,-2*12) from dual;

last_day():
used to get last day in the month

select last_day(sysdate) from dual;
30-jun-22

next_day():
used to find coming weekday date

syntax:
next_day(date,weekday)

next_day(sysdate,'fri')	returns next Friday date

months_between():
used to get no of months b/w 2 dates

syntax:
months_between(date1,date2)

calculate experience:

```
SELECT empno,ename,hiredate,
(sysdate-hiredate)/365 as experience
FROM emp;
(or)
SELECT empno,ename,hiredate,
trunc(months_between(sysdate,hiredate)/12) as
experience
FROM emp;
```

conversion functions:

to_char()
to_date()
to_number()

There are 2 types of conversions:

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

✓

- **Implicit Conversion:**
  if conversion is done implicitly by ORACLE then it is called "Implicit Conversion"

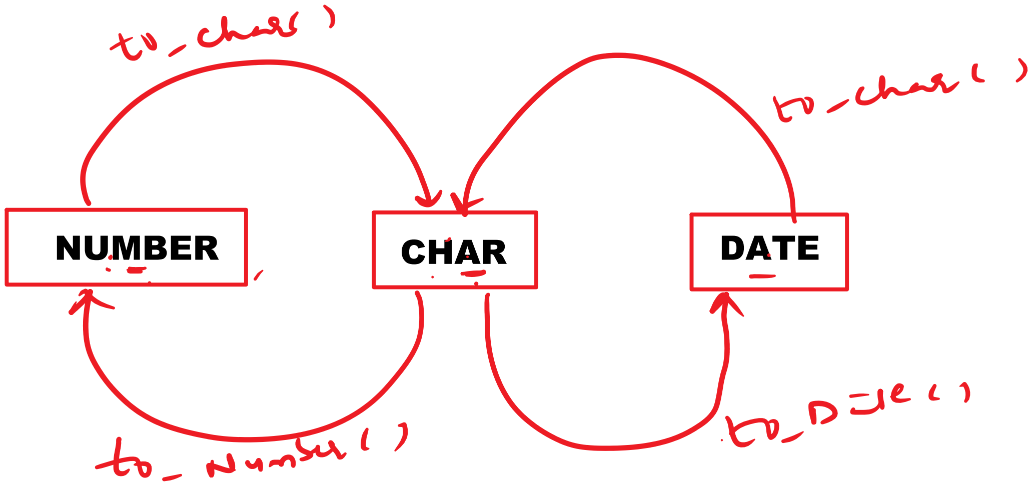
```
SELECT '100'+'200' FROM dual;
300
```

string string

num num

100 + 200 → 300 ✓

Explicit Conversion:
If conversion is done by built-in function then it is called "explicit conversion"



to_char() [date to string]:

- can be used to convert date value to string.
- using this we can change the date formats

Syntax:
to_Char(date,format)

dd-mon-rr

23-dec-19

us date format:
mm/dd/yyyy

ind date format:
dd/mm/yyyy

format	purpose	output
yyyy	8-jun-22 year 4 digits	2022
yy	year 2 digits	22
yyy	year 3 digits	022
y	year 1 digit	1
year / YEAR	year name in words	twenty twenty-two/ TWENTY TWENTY-TWO
mm	month 2 digits	06
mon	short month name	jun
month	full month name	june
d	day num in week	4
dd	day num in month	8
dy	short weekday name	wed

day	full weekday name	Wednesday
q	quarter num jan-mar => 1st qrtr apr-jun => 2nd qrtr jul-sep oct-dec	2
cc	century	21
hh/hh12		
hh24		
mi		
ss		
am / pm		

```
select to_char(sysdate,'hh:mi:ss am') from
dual;

select to_char(sysdate,'hh24:mi:ss ') from
dual;
```

display emp hiredates in us date format:

```
SELECT ename,sal,
to_Char(hiredate,'mm/dd/yyyy') FROM emp;
```

display emp hiredates in ind date format:

```
SELECT ename,sal,
to_Char(hiredate,'dd/mm/yyyy') FROM emp;
```