Unit-II

Chapter-2 The Basic (Flat) Relational Model

Most of the DB systems are constructed using relational model. Relation can be defined as a collection of flat records (or) a table containing data (a set of values) belonging to a physically or conceptual existence objects.

Relational model concepts

1. **Domain**: It is defined as set of possible atomic values They are identified by associating the data type

Domain — dom(Ai) — A – aadhaar no – Number (12)

Attribute ~~Relation~~: Describe the structure of relation

Relation: A relation is $R(A_1, A_2, \ldots, A_n)$ with a set of 'n' records or rows or tuples, where each Ai consists atomic value from dom(Ai), there each Ai is attribute

Tuples: Individual record / row in table

2. Characteristics of Relations:

1. Ordering among the tuples is no importance

records	sno	sname	class	branch
Tuples r_1 →	9181T055	XX	2	It
r_2 →	9181T001	YY	2	it
r_3 →	9181T045	XX	3	it

2. Ordering among the values of attribute in a tuple will have importance.

$R(A_1, A_2 -- A_n)$ — $t_1 = \{v_1, v_1, v_3, -- v_n\}$

$t_2 = \{v_1, v_2, -- v_n\}$

3. The values and null values - which inapplicable/unavailable to attribute of tuple.

4. Intersection of relation - easily identified.

8. Notations

i) Relations - we have R, S, T, U

ii) Relationship set - r, s, t, u

iii) Attribute with relation - $R.A_1$ (relation / Attribute name)

iv) Tuples - $t(r)$, $t(s)$, $t(t)$, $t(v)$

v) Attribute within a tuple - $t.A_i$ or $t[A_i]$

Relational model concepts:

1. Domains, Attribute, relation, tuple
2. Characteristics of relation
3. Notations

→ ER - to - Relational Mapping Algorithm:- In relational model database is collection of relations where in each relation information is recorded by giving a relation name 'R' with set of tuples as a relation state

$r = \{t_1, t_2, ---t_n\}$ and each tuple $t_i = \{v_1, v_2, ---v_n\}$ containing a value for v_i or a null value for each attribute A_i in $R(A_1, A_2 ---A_n)$. These relat Inorder to obtain these relations, consider ER model for a company database application in which we need to keep to track the details concerned to entities belonging to entity types called employee, department, project, and dependence.

(Re-draw ER-model of company database)

Apply the following steps in order to obtain set of relations:

Step-1 Mapping regular or strong entity type - For each regular entity type E in ER model construct a seperate relation or R with a set of attributes attatched to E. If E contains composite attribute then all its simple attributes are to be included into R. One of the attribute A_i are represent as key attribute by which a unique tuple can be extracted from that relation.

Here Employee

ssn	fname	initial	lname	dob	Address	sex	salary

Department

dno	dname	dloc

Project

pno	pname	ploc

Step-2 Mapping of weak entity type. For each weak entity type E create a new relation or R with its attributes an based on identifying relation ship identify owner entity type E_1 and add key attribute of E_1 as foreign key attribute with R

✓dependent

F.K				
essn	name	sex	Bdate	Relation

Once the relation is identified represent the combination of partial key and foreign key as key attribute in R

Step-3 Mapping of one-one (1:1) relation ship type - On a ER model if it contains a relation ship of with 1:1 cardinality constraint then identify 2 entity type S & T participating in that relation and entity with total participation assume in place of S and then add key attribute of Relation T as foreign key attribute into S.

any dept can't be without manager

department manages employee

department

			F.K
dno	dname	dloc	dmgrssn

Step-4: Mapping of one-to-many (1:N) or N:1 relation ship type: for each 1:N er model identify participating entity types S & T, entity type with n side participation assume in place of S and then add key attribute of T as a foreign key attribute with S.

1:N - works_for Employee department

Employee

ssn	Fname	initial	Lname	Bdate	Address	sex	salary	dno (F.K)

1:N - controls project dept

✓project

dno	dname	ploc	dno (F.K)

supervisor employee employee

✓Employee

ssn	fname	initial	lname	dob	address	sex	salary	dno (F.K)	superssn (F.K)

Step-5 Mapping many to many (N:N) Relation ship types: For each N:N RT in ER model create a new relation R with key attributes of participating entity types as foreign key along with if that relation ship contains any attribute. Combination of all the attributes represent as key attribute in R.

✓works_on

essn (F.K)	pno (F.K)	hours

Step-6 Mapping Relationship with attributes:- In ER model if any relation (contains) is attatched with a simple attribute then add that attribute to one of participating entity type with total participations.

department

dno	dname	dloc	dmgrssn	mgrstartdate

Step 7: Mapping multi-valued Attributes: If any entity type ~~contains~~ ER model contains multi valued attribute then remove that attribute from the created relation and construct a new relation

with that attribute along with key attribute of the relation from which it is removed. In a new relation combination of attributes is treated as primary key attribute for that relation.

✓ department

dno	dname	dmgrssn	mgrstartdate

/department_loc

F.K	
deptno	dloc

Step-8 Mapping Relation ship type with degree more than 2

If degree of relationship type is >2 then convert into binary relationship and then apply above rules.

The final set of relations for company database scheme are given as follows: ✓ entities

→ Constraints on Relational Database:

1. Implicit constraints or Model based constraints
2. Explicit constraints or Schema based constraints
3. Application programming constraints or Business rules

2. (a) Domain constraints
   (b) Key constraints
   (c) Entity Integrity constraints
   (d) Referential Integrity constraints

Domain constraint: States that if you are giving a value in tuple it must be atomic value and which is available in that domain.

Domain is identified by the datatype and size of datatype of that attribute.

Key constraint: i, Super key

ii, Primary key or key attribute

iii, Candidate key

iv, Primary key

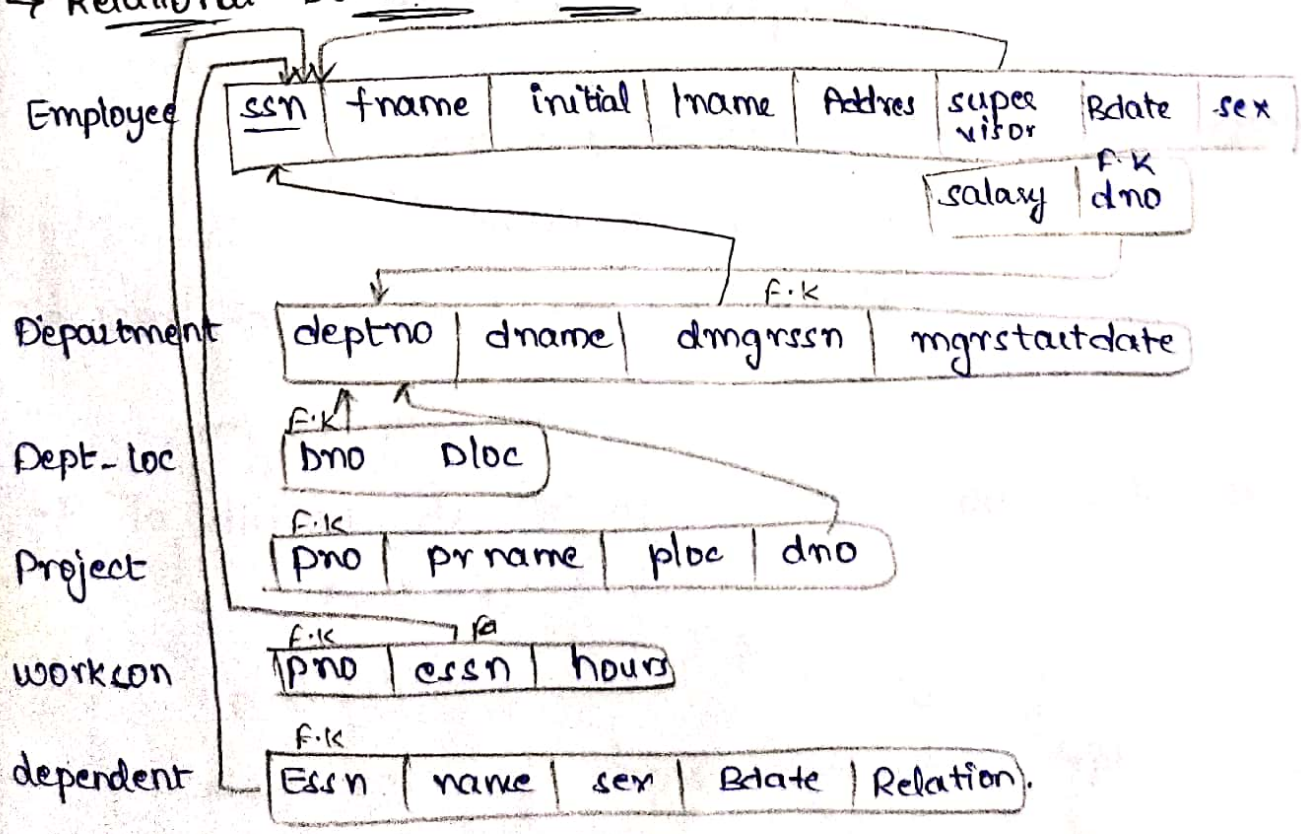
v, Secondar key

Entity Integrity : The primary key attribute cannot be set as null

→ Degree of Relation : No. of attributes attr available in that relation.

→ Degree of Relationship: No. of participating entity types.

→ Relational Database Schema:



Update operations & Dealing with constraint violations:

1. Insert
2. Delete
3. Update

Relation DB Query Language:

1. Relational Algebra
2. Relational Calculus

1. (a) Unary operators - SELECT, PROJECT, RENAME
   (b) Operations from set theory - UNION, INTERSECTION, MINUS
   (c) Binary operations
       - DIVISION operation
       - JOIN Operation - 1) Theta
         2) Equi
         3) Natural
         4) Outer - Left OJ
            Right OJ
            Full OJ
   (d) Operations with Aggregate functions

Select operation is used to retrieve all tuples ~~specified typ tuple~~ or subset of the tuple

$\sigma_{<condition>}(R)$

<condition> ↓

<attribute_name> rel-comp-op <value>

<attribute name> = <attribute_name>

→ WAQ to display details of employees

σ (Employee)

→ To display details of employee working for dept 1

σ (Employee)

(DNO = 1)

→ Display name of department whose dept = 1

PROJECT : Allows all the tuples of a relation for interested attributes

Π (R)
< attribute_name >

→ To display details of employee SSN & fname

Π (Employee)
SSN, Fname.

Rename: Change name of relation from one to another

ρ_S (R) (oldname)
< new_name >

$(\rho_S(R))$

→ To display ssn, fname, dno for all employees of dept 1

Π (ssn, fname, dept)

$\left(\sigma_{dno=1} (employee)\right)$

Sequence of Operations in Relational Algebra:

* Write a relational algebra query to retrieve fname, lname for all the employees working in deptno = 5

$$Dep5\_employees \leftarrow \sigma_{dno=5}(emp)$$

$$Result \leftarrow \pi_{fname, lname}(dep5\_employees)$$

(or)

$$R \leftarrow \pi_{fname, lname}(\sigma_{dno=5}(emp))$$

* And rename it

$$\rho_{(first\_name, last\_name)}(Result \leftarrow \pi_{fname, lname}(dep5\_employees))$$

→ Binary operations:

i) Set operations: ⓐ Union - ∪

ⓑ Intersection - ∩

ⓒ Difference - −

Write a query to display ssn's of employees working for dept 5 or supervising employees of dept 5

$$dep5\_emp \leftarrow \sigma_{dno=5}(emp)$$

$$R1\_SSN \leftarrow \pi_{ssn}(dep5\_emp)$$

$$R2\_mgrssn \leftarrow \pi_{superssn}(dep5\_emp)$$

$$Result \leftarrow R1\_SSN \cup R2\_mgrssN$$

* Write RAQ to display student and teacher first name

Student

fname	lname
XX	X
YY	Y
ZZ	Z

Teacher

f_name	l_name
XX	X
AA	A
ZZ	Z
BB	B

R_1_stu $\leftarrow \Pi_{fname}$ (student)

R_2_tea $\leftarrow \Pi_{f\_name}$ (teacher)

Result $\leftarrow$ R_1_stu $\cup$ R_2_tea

* Write RAQ to display whose first names are same:

Result1 $\leftarrow$ R_1_stu $\cap$ R_2_tea

* Write RAQ unique first name to teacher

Result2 $\leftarrow$ R_2_tea $-$ R_1_stu

Intersection: Let R, S be two relations

$R \cap S = ((R \cup S) - (R - S)) - (S - R)$

$R \cap S = (R \cup S) - ((R - S) \cup (S - R))$

ii, JOIN Operations:

① Employee:

ssn	fname	lname	sex	sal	dno	mgrssn
1234	XX	X	F	10000	5	1235
1235	YY	Y	M	20000	5	null

② Department

deptno	dname	dmgrssn	startdate
5	R	1235	10-dec-1985
1	A	1234	15-NOV-1986

③ Dependent

essn	name	Relation
1235	XXXX	Brother
1235	YYYY	sister

ⓐ Cross product

ⓑ Theta join

ⓒ Equi join

ⓓ Natural join

ⓔ Outer join

Cross join on two relations R and S is R x S contains 'mn' no. of rows from R containing m rows & S containing n no. of rows.

Cross join will not generate any meaningful result.

Result ← employee × department

Result contains 10 attributes & 4 rows.

Theta: join operation on two relations R & S is R ⋈ S
<join cond>

join condition is of the form cond1 or cond2 and cond

Ai θ Bj

Ai - attribute from R, Bj - attribute from S

θ - Any comparision operator

{ <, <=, >, >=, =, <> }

* Write a RAQ to display dept name for each employee

Result ← employee ⋈ department

dno = deptno

Result contains

ssn	fname	lname	sex	salary	dno	mgrssn	deptno	dname	dmgrssn	startdate
123A	XX	X	F	10000	5	1235	5	R	1235	10-dec-1985
1235	YY	Y	M	20000	5	null	5	R	1235	10-dec-1985

Result 1 ← Π ssn, fname, dno, dname (Result)

Equi: If we define theta join with '=' then equi.

Natural : Natural join on R,S is R * S.

It joins two relations which is common attribute and that attribute should be of same name and is displayed only one in result. If they are not of same name we have to rename.

Result ← employee * ρ (department(dno, dname, dmgrssn, sd)

Outer join operation:

@ Left outer join: emp ⟕ dept
dno = deptno

emp

ssn	fname	sal	dno
1234	XX	10000	1
1235	YY	15000	null

dept

deptno	dname
1	research
4	sales

output:

ssn	fname	sal	dno	deptno	dname
1234	XX	10000	1	1	research
1235	YY	15000	null	null	null

⑥ Right outer join: emp ⟖ dept
dno = deptno

ssn	fname	sal	dno	deptno	dname
1234	XX	10000	1	1	research
null	null	null	null	4	sales

(c) Full outer join: emp ⟗ dept
dno=deptno

ssn	fname	sal	dno	deptno	dname
1234	XX	10000	1	1	research
1235	YY	15000	null	null	null
null	null	null	null	4	sales

→ Group function in Relational Algebra:

$$\mathcal{F}_{\langle function\rangle}(R) \text{ or } \langle col\text{-}name\rangle$$

$$\mathcal{F}_{\langle func\rangle, \langle col\text{-}name\rangle}(R)$$

$$\langle grouping\text{-}attributes\rangle \mathcal{F}_{\langle aggregate\text{-}func\rangle}(R)$$

> To display total salaries dept wise

$$\mathcal{F}_{sum_{sal}, deptno}(employee)$$

complete set of RA operation:

{ σ, Π, ρ, ∪, −, × } with these operations we can form any type of complex query. in relational algebra.

- Division Operation: If R(X) and S(Y) are two relations, $Y \subseteq X$ and $R(X) \div S(Y)$ is a result T(Z) where Z = X-Y and t is a tuple in T holds $t_R(t) = t$ for each tuple t_S in S with same value there exists a tuple $t_R(t)$ in R.

R

A	B
a_1	b_1
a_2	b_1
a_3	b_1
a_4	b_2
a_1	b_4
a_2	b_4
a_3	b_4
a_4	b_5
a_1	b_1
a_2	b_2

S

A
a_1
a_2
a_3

Result T

B
b_1
b_4

RA to display fname & lname of all the employees working on project on which smith is working.

emp:

ssn	fname	lname	sal	deptno
12345	john	smith	15000	1
12346	franklin	Anna	10000	2
12347	lena	bez	15000	1

works_on

essn	pno
12345	10
12345	20
12346	10
12347	10
12347	20

smith_details ← $\sigma_{lname='smith'}$ (employee)

smith_projects ← smith_details ⋈ works_on
ssn = essn

pno
10
20

← smith_pno ← Π_{pno} (smith_projects)

essn
12345
12347

← temp_pno ← works_on ÷ smith_pno

$T_1 \leftarrow \rho_{T(ssn)}$ (temp_essn)

$T_2 \leftarrow T_1 * employee$

smith_colligues ← $\Pi_{fname, lname}$ (T_2)

→ Relational Calculus:

1. Tuple relational calculus (TRC)
2. Domain relation calculus (DRC)

① Syntax: $\{ t \mid cond(t) \}$

(tuple variables will range over a relation)

↓ formula

→ condition is ~~formula~~ of atom

↓

R(t) or S(r) → f_1 & f_2 are two formulas

$t.A_i = r.B_i$

$t.A_i = C$

$C = t.A_i$

i) f_1 or f_2

ii) f_1 and f_2

iii) Not f_1

→ existential quantifier ∃

→ universal quantifier ∀ are used in conditions.

RC	SQL	RA
$\{t \mid emp(t)\}$	select * from employee;	$\sigma(emp)$
$\{t \mid emp(t)$ and $t.fname = 'john'$ and $t.lname = 'smith'\}$	select * from employee where fname = 'john' and lname = 'smith';	σ (emp) fname = 'john' and lname = 'smith'

→ To display fname, lname, deptno and dname

$\{t.fname, t.lname, t.deptno, d.dname \mid emp(t), dept(d)$ and $t.deptno = d.dno\}$

② Domain RC:

emp → Display fname, lname of all employees.

u v w x y z a

ssn fname minit lname sal dno address

$\{v, x \mid emp(u, v, w, x, y, z, a)\}$

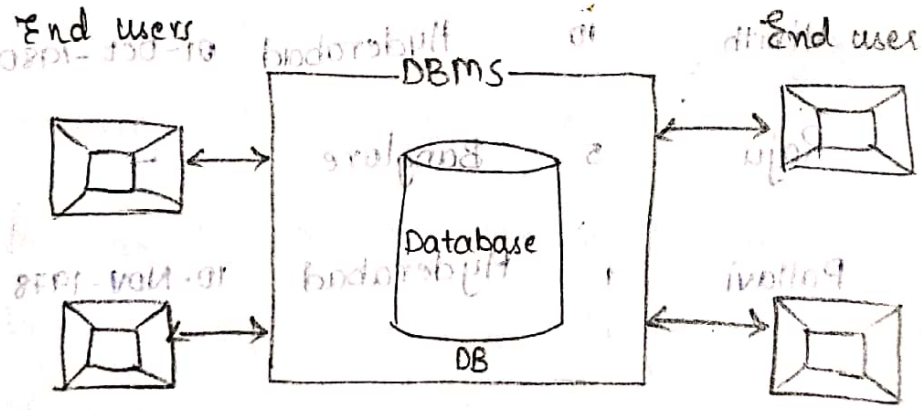
→ fname, lname of all employees working for dno = 1

$\{v\, x \mid emp(u, v, w, x, y, z, a)$ and $z = 1\}$

Sturctured Query Language (SQL)

* Database is a collection of inter-related data, the data or information is known facts which may be structured or un-structured nature. The collection of data must, to ensure logically coherent and has meaning.

* DBMS is a software which allows the users to define, construct, manipulate and maintain a database in a specific application. The simplified view of database system is represented as:



* In order to create a database Oracle corporation provided Oracle software which is a Relational Data Model of nature ORDBMS (Object-Oriented Relational Database Management System) installed with two components Oracle Server module & Oracle client module.

* SQL * PLUS is a package from Oracle acts as a client module which takes commands from the end user written english language statements by following SQL concepts.

* In RDBMS, the database is organised by using a collection of tables. Table is like a data repository in which data is organized by using rows and columns. Consider supplier data base which contains 3 database tables.

Supplier:

S.No	Sname	Status	City	SBDate
S1	Vinith	10	Hyderabad	01-Oct-1980
S2	Raju	5	Banglore	-
S3	Pallavi	1	Hyderabad	10-Nov-1978

Part:

PNo	PName	Pweight	Pcolor
P1	NUT	10	Brown
P2	Screw	5	white
P3	Bolt	15	red

Supply:

S.No	Pno	Quantity
S1	P1	100
S1	P2	150
S2	P1	200
S2	P3	100
S3	P1	1000

→ Data Types in SQL:

1. char(size) - This allows to hold a column of a database table with fixed no. of characters of specified size bytes. Maximum size is 2000 bytes. Default and minimum size is 1 byte. Ex: sname char(10).

2. Varchar2(size) - This allows variable length character string having maximum length size bytes. Maximum is 4000 bytes and minimum size is 1 byte.

Ex: sname varchar2(10)

if the size is 10 and enter name occupied is only 6 da bytes remaining 4 bytes are wasted if we 'char' data type no memory wastage when 'varchar' is used.

3. Number (size) - weight number (5)

Allows to store number natured data for a column of DB table.

4. Number (p,s) - Allows to store number data with precision p and scale s: Ex: 10555.75, 5000, 99952.50

5. date - this allows to store a valid date data along with time data also

Ex: SBDate date

6. timestamp - this allows to store a valid day, month year of a date data as well as hrs, min, sec of time

7. long - this allows to store character data of variable length upto 2GB Ex: files within 2GB

8. raw (size) - raw binary data of length size bytes maximum size is 2000B.

9. clob - character large object containing single byte characters maximum size is 4GB. Ex: files

10. blob - Binary large object maximum size is 4GB

→ SQL statements: SQL provides the following statements or commands during the construction and usage of database.

	Category	Commands or statements	Usage
1.	DDL (data definition Language)	create alter drop truncate rename	DDL statements are used for creating, replacing, altering, dropping, renaming database objects like tables, views, indexes...etc.
2.	DML (data manipulation Language)	insert delete update select	DML statements are used for inserting, deleting, modifying rows of a database table, retreiving rows of database table.
3.	TCL (Transaction control Language)	commit rollback save point	TCL statements allows guarantee database consistency and integrity by controlling the transactions using these commands.

4. DCL (data control Language)

grant
revoke

DCL statements are used for controlling access to the database and its objects.

1. Create Statement: This statement allows creating tables, views, like database objects. Here tables represents the logical container of a database and views represent holding data which is derived from other database tables. The following syntax is used to create table:

CREATE TABLE <TABLENAME>(column datatype constraints, ...)

Here tablename must be unique. Names of the column within the table must be unique. Specifying the constraints is optional. All SQL statements are not case sensitive. All table names are stored in database server always with uppercase only. If user wants to store tablename with lower case then enclose table name within double quotes

Creating supplier table:

SQL > create table Supplier(sno varchar(5),
sname varchar(25), status number(3),
city varchar(25), SBDate date); ↵

table create.

Creating parts table:

SQL > create table Parts (pno varchar(5),
pname varchar(10), pweight number(5),
pcolor varchar(10)); ↵

table created

Creating supply table:

SQL > create table Supply (sno varchar(5),
pno varchar(5), Quantity number(5)); ↵

table created

2. Insert Statement: This statement allows for organizing the data in a table row by row manner means it allows to insert row of a data into a table.

Syntax: 1. insert into <table name> values (value1, value2, - - - value n);

Note: The value to be inserted into the table if character nature or date nature enclose those values within single quotes.

Ex: insert into supplier values ('s1', 'vinith', 10, 'hyderabad', '10-OCT-1980');↵

Syntax 2: Inserting values to specific columns.

insert into <tablename> (col1, col2...)

values (value1, value2, ...)

Ex: insert into supplier (sno, sname) values ('s10', 'john');

Syntax 3: General insert statement to insert rows interactively (by using a single insert statement, making columns as substitution variables

Ex: SQL> insert into supplier values ('&sno', '&sname', &status, '&city', '&SDDate');↵

Enter value for sno: s1↵

Enter value for sname: vinith↵

Enter value for status: 10↵

Enter values for city: hyderabad↵

Enter value for SDDate: 10-NOV-1980↵

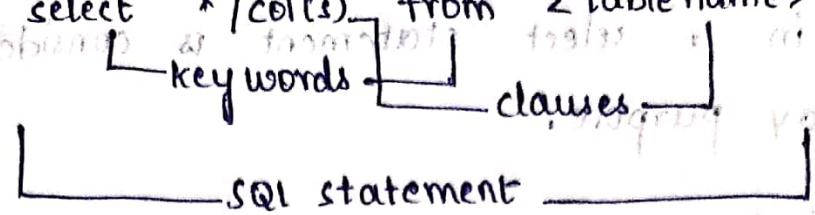
1 row created

SQL> / ↵ (to insert 2nd row)

3. Select statement: Select statement is a special kind of DML statement used to retrieve or extract data from a database table. select Basic select statement will contain 2 clauses select clause & from

clause:

Syntax: select */col(s) from <tablename>



An SQL statement contains a combination of 2 or more clauses which is split accross multiple lines Each select statement must be ended with ';'.

Select statement is used to retreive data for specific or all columns ~~for~~ of database table, sometimes in a select clause we may specify expressions also

Ex: Display details of all suppliers

```
SQL> select *
     from supplier;
```

SNO	sname	status	city	SBDate
S1.	.....	.....	....	
S2				
S3	.....	.....	....	
:				

Ex: To display sno along with status

```
SQL> select sno, status
     from supplier;
```

Column alias: names : The given alias names to the columns in a select statement is considered only for display purpose.

```
SQL > select sno as Supplier_No, status
      from supplier
```

Supplier_No	status
S1	10
S2	15
S3	5
!	

to specify 'as' in assigning alias name is optional.

Use of Arithmatic Operators & Relational comparision operators :-

+ → Addition, * → multiply, - → subtraction.
/ → division

Relational : < , < = , > , > = , = (equal), < > → not equal
(< > or ! or ^)

Select statement with WHERE clause: Select statement can be issued to retreive rows based on a condition from db table by associating a simple select statement with

where clause: Where clause is useful to restrict the no. of rows to be displayed based on the given condition.

Syntax: select *|col(s)

from <tablename>

where condition(s)

↳ colname relop value or colname

Ex: Write a query to display names of all suppliers whose status is < 5

SQL> select sname

from suppliers

where status < 5;

Write a query to display supplier no, quantity supplied, 10 % of increase in quantity to be supplied for next order for all suppliers.

SQL> select sno, quantity as current_supplied_value,

quantity + quantity * 0.10 as new_quantity_supplied

From supply;

expression

sno	current_supplied_value	new_quantity_supplied
S1	...	...
...	...	...

* To watch the contents in table, Enter

desc <tablename>

used to check names of columns in table.

Ex: desc student

* To view the names of database tables in a user issue following command

select * from tab;

→ Use of distinct keywords:

distinct keyword is used before the column in a select clause to remove redundant values from the retrieved results or values.

- Write a query (Waq) to display various branches in which students are studying.

SQL> select unique/distinct sbranch

from student

ORDER BY Clause: This clause instructs the DBMS to display retreived rows in a order based on specified column value. The data can be ordered either in ascending or descending.

Syntax: SQL> select */[columns

from <tablename>

where condition(s)

order by <column name> asc/desc;

By default order is ascending order. If explicity user wants to display in descending 'desc' keyword should be specified after the column. In any sql statement 'order by' clause must be last.

- Waq details of the student in an order based on fee paid

```
SQL > select *
      from student
      order by sfee;
```

- Waq display student details in it branch acc to fee paid in an descending order.

```
SQL > select *
      from student
      where sbranch = it
      order by sfee desc;
```

With order clause we can use column no also in place of name of a column.

* Value given for comparision in the where clause for comparision is case sensitive.

* NULL Value: Null value is a value which is unavailable, inapplicable or unknown value for a column of a row in a database table. Null value is not same as 'zero'.

- Waq to display details of suppliers whose birth date is not available.

SQL > select *
from supplier
where sDOB is null;

- Waq to display details of suppliers to whom birth date is available.

SQL> select *
from supplier
where sDOB is not null;

* Special comparision operations:

1. Between <lower value> and <higher value> - this operator is used to retreive the rows of a database table based on the value of a column which lies between inclusion of the given lower value and higher value.

- WAQ to display the student details who have paid fee between 50,000 and 100,000

SQL> select *
from student
where sfee between 50000 and 100000;

2. IN operator - This operator is used to check value of a column in a row retrieved by DBMS is in a list of values provided by the user.

- WAQ to display details of the students whose branch is IT or CSE.

```
SQL> select *
  from student
  where branch in ('cse', 'it');
```

3. LIKE operator: Like operator is a pattern matching comparision operator which refines rows retrieved by DBMS by using two wildcard patters called % and _ (underscore). Modulus symbols instructs compaision of 0 or more no.of characters in a sequence. Underscore represents occurence of a single character for comparision

- WAQ to display student names whose name start with p.

```
SQL> select *
  from student
  where sname like 'p%';
```

- WAQ to display names of the student whose name ends with a

```
SQL> select *
  from student
  where sname like '%a';
```

- WAQ to display name whose second character is a & 5th character is a.

```
select *
from student
where sname like '_a__a%';
```

4. Null operator: Null operator is used to check the existence of a null value for a specified column.

* Logical Comparision operator: In writing a query we can use the following logical comparision operator in order to form a single condition in a where clause if data is to be retrieved based on more than one condition

1. AND → used to retrieve data based on logical and
2. OR
2. NOT

- WAQ to display names of the students who are studiying in 'it' and paid fee more than 50,000.

```
SQL> select sname
from student
where sbranch = 'it' AND sfee > 50000;
```

- WAQ to display details of the students, who have paid fee less than 20,000 or paid more than 50,000.

```
SQL> select *
     from student
     where sfee < 20000 or sfee > 50000;
```

→ Constraints in SQL:

Constraints are rules/checks specified on a column of database table, which are automatically imposed whenever a user issues inserting values into a columns deleting rows or updating values of a columns in database table. to the DBMS. Here during these DML operations if data is inaccordance with specified constraints then operations will be performed otherwise if constraints are violated simply rejects the issued DML operation. The following are various constraints we can specify on a column of database table.

1. NULL / NOT NULL
2. UNIQUE
3. CHECK
4. PRIMARY KEY
5. FOREIGN KEY

1. NULL / NOT NULL: By default during the database table creation the DBMS will add a NULL constraint to all the columns in a table. If explicitly user wants to make a column of a table to have

any value then that is specified through NOT NULL constraint.

```
SQL> create table employee (eno varchar2(4), ename
     varchar2(20) NOT NULL, ejob varchar2(15));
Table created.

SQL> insert into employee values('e111', 'smith', 'analyst');
1 row created.

SQL> insert into employee (emp, ename) values ('e211', ' ');
     constraint violation error.

SQL> insert into employee (eno, ename) values ('e211',
     'smith');
1 row created.
```

2. UNIQUE: This constraint allows a column to have a unique or distinct value. A column with unique constraint accepts a null value also. We can specify one or more columns as a combination with unique nature also. In this case no two rows will have a same value for such combination.

```
Ex: SQL> create table student (sno varchar2(10),
         unique, sname varchar2(10));
table created.

SQL> insert into student values ('yisit001',
     'madhuri');
1 row created.
```

insert into student values ('yisit001', 'madhu'); ← 1 row created

SQL > insert into student values ('yisitoo1', 'madhu');

constraint violation error.

SQL > insert into student values (' ', 'madhuu');

1 row created.

SQL > create table student (sno varchar2(10), sname varchar2(10), UNIQUE (sno, sname));

SQL > insert into student values ('yisitoo1', 'madhu');

1 row created.

SQL > insert into student values ('yisitoo1', 'vaishu');

1 row created

Since 'yisitoo1 & madhu' combination is not repeated in second row.

3. CHECK: This constraint allows to check a value given for a column is according to the values supplied for that column with check constraint during creation of database table. If violated DBMS simply rejects the operation.

Ex: SQL > create table student (sno, varchar2(10), sfee number(10), check (sfee > 50000), class number(5), check (class in (1, 2, 3, 4));

table created.

SQL > insert into student values ('yisitoo1', 40000, 05);

constraint violation error

SQL> insert into student values ('y18it001', 107000,

constraint violation error.

SQL> insert into student values ('y18it001', 107000,

1 row created.

4. PRIMARY KEY: A column of a database table is specified with primary key constraint then it makes a column as to accept a value with 'not null' & 'unique' nature. Means primary key constraint that specifies a column of a table as not null & unique. Primary key column also allows to retrieve a single ~~row~~ unique row from a set of rows in a table if specified column with that constraint in condition filled of 'where' clause,

A table atmost have only one primary key.

Ex: SQL> Create table student (sno varchar2(10) primary key, sname varchar2(10), branch varchar2(10)

SQL> insert into student values ('y18it00', 'madhu', 'it');

1 row created.

SQL> insert into student values ('y18it002', 'vaishu', 'it');

1 row created.

SQL> insert into student values ('y18it001', 'vaishu', 'it');

constraint violation error.

5. FOREIGN key: Foreign key constraint is specified on a column(s) of a database table if a column references a value from the column of another db table in which that column is made as primary key nature. The table which references a value of a column is set as referencing table and the table which supplies a value is called referenced table. For this first create referenced relation, insert rows then create referencing relation, after insert rows.

Ex: Branch: (Referenced Table)

P.K Branchid	branchname	branchHOD
it	into-tech	sri krishna
cse	comp science	sri latha

Student: (Referencing Table)

P.K sno	sname	class	F.K branchid
y18it001	madhu	2	it
y16it001	pavani	4	it
y15cs001	naishu	2	cse

SQL> create table branch (Branchid varchar2(10) primary key, branchname varchar2(10), branchHOD varchar2(10));

table created.

→ Insert Data into Branch table

SQL> create table student(sno varchar2(10) primary key, sname varchar2(10), class number(1), Branchid varchar2(10) FOREIGN KEY REFERENCES Branch(Brid));

Insert Student Records.

SQL> insert into student values('y15cs001', 'madhu', 2, 'ec');

constraint violation error

Since 'ec' value is not present in Branch(Brid) column.

Syntax: DELETE from <table name>
where condition(s);

→ To delete all the rows from employee

SQL> delete from employee;

SQL> select * from employee;

SQL> Rollback;

SQL> select * from employee;

→ To delete a row from employee with empno 7369

SQL> Delete from employee

where empo = '7369';

→ To delete all row from employee with Dno = 10;

SQL> Delete from employee

where deptno = '10';

→ To illustrate savepoint

SQL > delete from employee

where empno = 7369;

SQL> savepoint s1;

SQL> Delete from employee where deptno = 10;

SQL> insert into employee (empo, empname, deptno) values

(1111, 'xxx', 20);

→ UPDATE : Update statement is used to modify a value of a column in a row or for a value for column in all the rows.

This statement also allows to modify a values for more than 1 column also.

Syntax: update <tablename> set [column = value],
[column = value];
where condition(s);

If where class is not used modification is done to all the rows of a database table for a specified column.

→ To update comm with 1000 for all employees

```
SQL> update employee set comm = 1000;
SQL> select * from employee;
SQL> Rollback;
SQL> select * from employee;
```

→ To update comm with 1000 for empno = 7369;

```
SQL> update employee set comm = 1000,
        where empno = 7369;
SQL> select * from employee;
```

→ To update comm with 1000 & sal with 10% increment for employee no = 7369

```
SQL> update employee set comm = 1000, sal = sal + sal *0.1
        where employeeno = 7369
```

→ COMMIT: commit statement causes dbms to make changes done through insert, delete & update into db table permanently.

→ RENAME: Rename statement causes dbms to change the name of a database table from old name to new name

Syntax: Rename <tablename> to <new table name>;

→ DROP: This statement cause dbms to remove permanently the rows & structure of a db table in a db. DROP statement cannot be rolled back.

Syntax:

DROP table <tablename>;

SQL> drop table employee;

→ TRUNCATE: Truncate statement instructs dbms to remove all the rows from a db table. This statement cannot be rolled back. This statement deletes only rows but structure of table remains unchanged

Syntax: Truncate table <table name>;

SQL> Truncate table employee;

Note: The following statement can be used to create a db table by copying all the rows from a table along with structure which is available in another user account.

Syntax: create table <table name> As (select * from <existing table name in another account>);

SQL> create table emp as select * from NIBITOSH.employee;

→ ALTER: Alter statement is used to modify the structure of database table.

* It allows to add a new column to database table.

* It allows to modify the datatype for particular column from one datatype to another datatype, if no rows contains a value for such column.

* It allows to change length or no.of bytes assigned for a column

* It allows to add any constraints such as primary key, not null, foreign key etc., to a columns of db table.

* This statement also allows to drop or remove a particular db table

Syntax: ALTER table <table name>

/add [colname datatype]

/modify [colname newdatatype]/

[colname datatype with size]

/drop column <columnname>/;

→ To add a column

SQL> alter table employee add hiredate date;

→ To modify a datatype

SQL> alter table employee modify hiredate date;

→ To modify bytes

SQL> alter table employee modify ename varchar2(1

→ To drop a column

SQL> Altel table employee drop column hiredate;

→ Syntax to add constraint

altel table <table name> add constraint

<constraint_name> constraint type;

To add primary key constraint using altel

SQL> altel table employee add constraint c1

primary key (empno);

SQL> altel table department add constraint c2

primary key (dno);

To add foreign key constraint

SQL> altel table employee add constraint c3

foreign key (deptno) references department (dno);

SQL> altel table employee add constraint c4

foreign key (mgr) references employee(eno);

SQL FUNCTIONS :-

→ Functions - built in functions provided with oracle allows to use in any SQL statements which can perform

1. Complex calculations and data
2. Modify values of data elements/columns
3. Can operate on group of rows to modify the resultant value.
4. To alter the date formats for display purpose.

These functions are catagorized into 2 types

1. Single row functions- which will operate on a single row of a database table and returns results produced by function.
   - (a) Number functions-
   - (b) Character functions
   - (c) Date-time functions
   - (d) Conversion functions
   - (e) General functions

2. Aggregate Functions or Group Functions: Group functions, these functions are applied on a group of rows based on supplied argument, processes and displays the results

(a) Number Functions: i, ABS(n) - this function returns the magnitude absolute value of n

```
SQL> select abs(-85), abs(85)
     from dual;
```

For dual is a dummi table created in oracle upon installing to be used in query whenever user supplies actual value of an argument for a function directly.

ii, ceil(n) - this function will return a smallest integer which is >= a supplied value n.

```
SQL > select ceil(-87.3), ceil(87.3), ceil(87)
      from dual;
```

iii) floor(n) - This function will return a largest integer which is ≤ given value n.

```
SQL> select floor(-87.3), floor(87.3), floor(87)
     from dual;
```

iv) MOD(m,n) - This function returns remainder m divided with n.

```
SQL> select mod(15, 4)
     from dual;
```

3

v) power(m, n) - This function returns m raised to the power(n) of n.

```
SQL> select power(3,4), power(3.5, 4.5), power(103, 1/3)
     from dual;
```

vi) sign(n) - This function will return -1 if n is negative and 1 if n is positive and 0 if n is zero.

```
SQL> select sign(-5), sign(5), sign(0)
     from dual;
SQL> select sign(1500 - salary)
     from employee;
```

vii) sqrt(n) - This function will return sqrt(n) if n is null it returns null if n is 0 it returns 0 if n is negative it returns error value

```
SQL> select sqrt(9), sqrt(0), sqrt(20)
     from dual;
```

(iii) Round (m, (n)) - This function rounds m to the n decimal places if n is omitted it is rounded to no decimal places, if n is negative numbers left to the decimal point are rounded.

```
Sql > select round (91.783), round (91.783, 1),
      round (91.783, -1)
      from dual;
```

91 , 91.8, 90

(b) Character Functions : (i) Character functions returning character values

1) CHR (x) - this function will return the character that has the character equivalent to.

```
SQL> select CHR (97)
     from dual;

SQL> select CHR (sal)
     from employee;
```

2) CONCAT (string1, string2) - this function returns the concanterated string from string1, string2

```
SQL> select concat ('RVRJC', 'college of eng')
     from dual;

SQL> select concat (empno, ename)
     from employee;
```

Note: If the contatenation is made for more 2 strings then use pipe characters (||)

```
SQL> select 'RVR' || 'JC' || 'college of eng'
     from dual;
SQL> select empno || ename || ' ' || sal
     from employee;
SQL> select empno || ' ' || ename || ' earns salary '
     || sal.
     from employee;
```

3) INITCAP (string) - this function returns initial letter as a capital for the string.

```
SQL> select initcap(ename)
     from employee;
```

4) LOWER (string) - this function converts the given string to lower case

```
SQL> select lower(ename).
     from employee;
```

5) UPPER (string) - this function converts the given string to upper case

6) LPAD (string, n) - this function pads the given string value starting from left to given n.

7) RPAD (string, n) - this function pads the given string value, starting from right to given n.

```
SQL> select LPAD(ename, 10)
     from employee;
SQL> select RPAD(ename, 10)
     from employee;
```

ii) Character functions returning Number values:-

1) ASCII(string) - this function returns the ascii value of first byte of the string in a database character set.

```
SQL> select ascii('A'), ascii('a'), ascii('Anil')
     From dual;

SQL> select ascii(ename)
     from employee;
```

2) instr(string, substring) - this function returns an integer indicating position of substring in the given string.

```
SQL> select instr('RVRJCCE','N')
     from dual;
SQL> select ename, instr(Ename,'a')
     from employee;
SQL> select length
```

3) LENGTH(string) - this function returns no. of characters in string

```
SQL> select length('RVRJCCE')
     from dual;
SQL> select ename, length(ename)
     from employee;
```

⑥ Date-time functions:

1. sysdate - it is an argument less function which returns the current date and time in client zone/terminal

```
SQL> select sysdate
     from dual;
```

2. systimestamp - this function will also returns date, time along with fractional seconds in database server zone.

3. Add_months (date, n) - this function add n no. of calendar months to the given date.

```
SQL> select Add_months (sysdate, 4)
     from dual;
```

```
SQL> select ename, hiredate, Add_months
     (hiredate, 2)
     from employee;
```

4. months_between (date1, date2) - this function returns the no. of months between specified dates.

```
SQL> select months_between (sysdate, hiredate)
     from employee;
```

5. Last_day (date) - this function returns last day of specified date.

```
SQL> select sysdate, last_day (sysdate)
     from dual;
```

→ 31-12-2019

```
SQL> select hiredate, last_day (hiredate) from
     employee;
```

6. round (date, format element) - this function rounds the supplied date based on supplied format element

```
SQL> select round(sysdate), round(sysdate,'month')
     , round(sysdate,'year')
     from dual;
```

Note: The arithmatic operations is also allowed on dates.

1. date + number - this function adds the no. of dates to given data and displays new date as result.

```
SQL> select sysdate, sysdate+10
     from dual;
```

2. date - number : subtracts no. of days from given date.

```
SQL> select sysdate-10
     from dual;
```

3. date1 - date2 : this returns the no. of days as result between given dates.

```
SQL> select sysdate - sysdate
     from dual;
```

Write a query to display no. of days an employee working with organization

```
SQL> select ename, hiredate, sysdate-hiredate
     as days
     from employee;
```

if we want no. of years

```
SQL> select ename, hiredate, round((sysdate-
     hiredate)/365) As exp_years
     from employee;
```

(d) Conversion functions: Oracle supports for explicit type conversion by using the following 3 conversion functions

1. to_char()
2. to_number()
3. to_date()

1. to_char() → this function converts the number or a date value to a varchar2 character string with format model specified by user.

format model element for number:

value	format element	result
4567	$999999	$4567
4567	0999999	004567

```
SQL> select to_char(5$457, $999999)
     from dual;
SQL> select ename, to_char(sal, $999999)
     from employee;
```

format element for date:

format element	meaning
MM	months in two digits
RM	roman numeral month
MON	month with 3 letter abbrevation
MONTH	fully spelled month

DD	two digit day of the month
DY	three letter abbrevated day.
DAY	full spelled day
W	week of the month
YYYY	year in 4 digits
YEAR	fully spelled year
HH	hours of day
MI	minutes of day
SS	seconds
TH	ordinal number
AM/PM	time indicator

```
SQL> select sysdate, to_char(sysdate,'MONTH/DAY/
     YEAR')
     from dual;
```

2. to_number(): this function converts a string representing varchar2 to a number nature

```
SQL> select to_number('+2345')
     from dual;
```

3. to_date() :- this function converts a varchar2 date value to date nature.

```
SQL> select to_date('12-dec-2019')
     from dual;
```

© General Functions:

1. NVL (exp1, exp2):-

This function assigns a non null value given through exp2 to the null value or a column containing a null value.

→ WAQ to display yearly salary for each employee along with comission.

```
SQL> select ename,(sal *12) + comm as yearly salary
     from employee;
```

```
SQL> select ename, (sal * 12) + NVL(comm,0) as yearly
     salary from employee;
```

2. Decode (exp (column, search, result, [search2], [result2], ..., default):-

It searches a value given in search field with a set of values available in exp(column, if match is found then the appropriate operations given in result are performed, if no match found then it applies operation given in default field. It works like case structure available in high level programming languages.

→ WAQ to display ename, existing salary, proposed salary based on if employee is a salesman apply 20% increment, if analyst apply 30% increment, otherwise only 10% increment.

```
SQL> select ename, sal as exst_sal,
        decode (ejob, 'salesman', (sal + (sal * 0.20)),
                'Analyst', (sal + (sal * 0.30)),
                'manager', (sal + (sal * 0.40)),
                (sal + (sal * 0.10))) as
        proposed_salary
    From employee;
```

3. Greatest (exp1, exp2 ---- expn) :-

It returns greatest value from supplied value/ expressions

```
SQL> select greatest (10, 5, 58, 99, 53) from dual;
SQL> select greatest ('A', 'a') from dual;
```

4. Least (exp1, exp2, ---- expn) :-

It returns least value from supplied value/from supplied expressions.

```
SQL> select least (eno, esal)
        from employee;
SQL> select least ('A', 'a')
        from duals
```

5. Vsize(exp):-

It returns total no.of bytes occupied for internal representation of supplied arguments/exp.

```
SQL> select vsize(14), vsize('Anand'), vsize('12-09-2019')
     from dual;

SQL> select ename, vsize(ename) as no.of_bytes_occupied
     from employee;
```

6. User:-

It displays the name of the current session user.

```
SQL> select user
     from dual;
```

II

Group functions (Aggregate functions)

It operates on a group of rows based on the columns supplied and applies the specified function and generates a single value.

1. SUM (DISTINCT/ALL exp/column name):

It takes a values available from the rows of a DB table from the supplied column name and performs the summation and displays the obtained result as output.

Ex: to display total amount paid to all employees

* If 'All' is preceded by column name then it considers duplicate values also for summation, by default sum function considers all.

* If 'distinct' it will restrict redundant or duplicate values to be considered for summations.

SQL> select sum(sal), sum(All, sal), sum(distinct, sal
from employee;

(a) Group By clause and having clause :-

The general SQL statement syntax

SQL> select
* |col1, col2, --- coln | Aggregate functions
from < table name(s)>
where < condition(s)>
group by < col name >
having < condition(s)' on aggregate functions >

Groupby clause if used in a statement along with a column name then group by clause partitions the no. of rows in a DB table into these many partitions based on the no. of distinct values available for that column and then applies the specified ~~argument~~ aggregate function in the select clause.

→ WAQ to display summation of the salaries paid department wise.

```
SQL> select deptno, sum(sal) as total_salary
     from employee
     group by deptno;
```

O/P

deptno	total_salary
10	value1
20	value2
30	value3

Having clause is used in SQL statement to refine the results written by aggregate functions along with group by clause.

→ WAQ to display the deptno to which to which salary paid is more than 10,000

```
select deptno, sum(sal) as total_salary
From employee
groupby deptno
having sum(sal) > 10,000;
```

deptno	total_salary
20	12575
30	15000

There is no ~~aggregate~~ ~~fun~~ having clause without aggregate function.

→ WAQ to display total salaries paid job categ wise

```
SQL> select job, sum(sal) as total_salary
     from employee
     group by job;
```

(b) MAX [Distinct| all exp|column]:-

It returns the maximum value from set of values available for a specified column

WAQ to display the maximum salary paid in an organisation.

```
SQL> select max(sal)
     from employee;
```

WAQ to display maximum salary paid deptwise

```
SQL> select deptno, max(sal)
     from employee
     group by deptno;
```

WAQ to display the deptno with highest salary if the highest salary is >3000

```
SQL> select deptno, max(sal)
     from employee
     group by deptno
     having max(sal) > 3000;
```

© min (distinct|all expl column):-

It returns a minimum value from the set of values available in the specified column.

```
SQL> select job, min(sal)
     from employee
     group by job;
```

WAQ to display job category within a dept to which min sal is paid. < 1000

```
SQL> select job, deptno, min(sal)
     from employee
     group by job, deptno;
```

ⓓ Avg (distinct|all expl column) :-

It returns average value obtained from a set of values available for the given column (set of values avail)

WAQ to display average salary paid by considering all the employees

```
SQL> select avg(sal)
     from employee;
```

WAQ to display average salaries deptno wise

```
SQL> select avg(sal)
     from employee
     group by deptno;
```

(8) count (* / columns with primary key):-

It counts the no.of rows available within a DB table

WAQ to display total no.of employees available within the organization

```
SQL> select count(*)
     from employee;

SQL> select count(empno)
     from employee;

SQL> select count(comm)
     from employee;
```

WAQ to display total no.of employees deptwise

```
SQL> select deptno, count(*)
     from employee
     group by deptno;
```

Advanced Query Statements

1. Set operators : SQL also supports the set operators called UNION, UNION ALL, INTERSECT, MINUS, as in mathematical set operations like union, intersection, set difference

i) UNION - union operation combines the rows returned by individual query in a component query. Union operation will remove the duplicate rows returned by individual query inorder to display final result.

Note: When using set operations the columns specified in each individual queries must be to satisfy the data compatability means columns must to have same data type.

Write a query to display job titles for department no : 10 or 20

```
SQL > select job from employee
      where deptno = '10'
      UNION
      select job from employee
      where deptno = '20';
```

ii) UNION ALL : It is similar to union but will not suppress duplicate rows from the result set.

Write a query to display all job titles in deptno = 10 or 20.

```
SQL> select job from employee
     where dept = 10
     UNION ALL
     select job from employee
     where dept = 20;
```

iii, INTERSECT: This operator is used to combine the rows return by each individual query and to display the rows which are common in each individual query.

Ex: Write a query to display the common job title available in deptno = 10 & 20

```
SQL> Select job from employee
     where deptno = '10'
     INTERSECT
     Select job from employee
     where deptno = '20';
```

iv, MINUS: This operator is used to combine rows return by individual query and to display rows which are unique to first query in component query.

Ex: Write a Query to display a unique job titles in department 10 compared to 20.

```
SQL> select job from employee
```

where deptno = '10'

MINUS

select job from employee

where deptno = '20';

Write a Query to display unique job titles in department no 10 in comparision with 20 and common with department 30

SQL> (select job from employee

where deptno = '10'

MINUS

select job from employee

where deptno = '20')

INTERSECT

select job from employee

where deptno = '30';

2. Retrieving Data from more than one table using JOINS:

Inorder to retrelve the data from more than one database tables SQL allows to perform various joins operations

(i) cross join

(ii) equi join or simple join or inner join

(iii) non-equi join

(iv) outer join

(v) self join

i, Cross join :- This kind of join performs the cartesian product on the rows available in a tables specified in the 'from' clause. Suppose in from clause 2 tables, T_1 and T_2 are specified T_1 contains 'm' no.of rows and T_2 contains 'n' no.of rows then result set after cross join will contain mxn no.of rows by associating each row of table T_1 with every row of T_2. The following query can perform cross join operation but the result generated is meaningless. those meaningless rows are called 'spurious' rows.

```
SQL> select *
     from employee, dept;
```

ii, equi join :- Equi join is used to generate a meaning-ful data upon performing the join operation. In equi join the join condition is defined in a 'where' clause on the columns of a table on which join can be performed by using '=' operator. Hence it is called equi join. The join condition may be of the form T_1.col_name = T_2.colname where T_1 & T_2 are tables specified from a from clause and colnames are on which equal comparision can be made, most the time join condition is defined on primary key of one table with foreign key of another table.

WAQ to display deptnames for all the employees

SQL> select * from employee, department
where employee.deptno = deptartment.dno;

with alias name →

SQL> select * from employee e, department d
where e.deptno = d.dno;

iii, non-equi join: In order to retreive the data from more than one database table if join condition is defined other than equal operator then such joins are called non-equi joins.

WAQ to display sal_grade for each employee based on salary of an employee.

SQL> select employee.ename, employee.sal, salgrade.grade
from employee, salgrade as salary_grade
where employee.sal between salgrade.losal and
salgrade.hisal;

WAQ to display deptname for employee along with grades based on salary.

SQL> select employee.ename, employee.sal, employee.deptno,
department.dname, salgrade.sal grade
from employee, department, salgrade
where employee.deptno = department.deptno and
employee.sal between salgrade.losal and
salgrade.hisal;

iv, Outerjoin: Outerjoin is used to display the matched rows from two tables based on join condition along with unmatched rows from the database table. Here outerjoin operator '(+)' is used to specify after the column of a table name in join condition with a nature of defecient table.

WAQ to display the result of outer join operation

```
SQL> select *
     from employee, department
     where employee.deptno(+) = department.deptno;
```

WAQ to display the department name which do not have no employee

```
SQL> select *
     from employee, department
     where employee.deptno(+) = department.deptno
     and employee.eno is null;
```

Outerjoin are of 3 types left outer join, right outer join and full outer join

Left outer join: this join is used to display all the matched rows based on given join condition along with unmatched rows from the left table side relation

Syntax:

```
select *
from table1
     left outer join table2
     on table1.col-name = table2.col-name;
```

Right outer join: This kind of join is used to display all the matched rows from join condition along with unmatched rows from right side relation is called right outer join.

Syntax: select *
from table1
Right outer join table2
on table1.col_name = table2.col_name;

Full outer join: This kind of join is used to display all the matched rows from join based on join condition along with unmatched rows from both the tables.

Syntax: select *
from table1
full outer join table2
on table1.col_name = table2.col_name;

iv, Self join: (or) Recursive join: If join condition is specified based on primary key and foreign key columns of same database table then such join is called self join.

WAQ to display employee number, employee name, manager number, manager name for all the employees.

SQL> select e.no, ename, mgr, empno as mgr_name
from employee e, employee m
where e.mgr = m.empno;

Sub Queries

Sub query: If a select statement nested within another select statement and produces the intermediate value as a result then such query is called sub-query.

In subquery, the query is written by writing outer query and inner query. The inner query must be enclosed within a paranthesis after the comparision operator, in an condition a field after 'where' clause or with 'having' clause. The result produced by the inner query will be taken as comparision value by the outer query in order to display the result.

Syntax:

select */columnlist

from <table name>

where colname comparision operator (select */columnli

from <table name

where <condition>

group by <colum

having <condition>

There are two kinds of subqueries:

1. Single - Row subqueries
2. Multiple - Row subqueries

1. Single Row subquery: If the inner query returns single row or single value as result then such queries are called single-row subqueries.

With single-row subquery we can use always any relation comparision operator in an condition field in outer query.

→ WAQ to display the details of employee who is earning highest salary within the organisation

```
SQL> select * from employee
     where sal = (select max(sal)
                  from employee);
```

→ WAQ to retrieve colleques of an employee "Miller".

```
SQL> select *
     from employee
     where deptno = (select deptno
                     from employee
                     where ename = "MILLER"
                     AND ENAME<>'MILLER');
```

↳ to delete miller row

→ WAQ to display the names of employees whose salary is less than the salary of "ford".

```
SQL> select ename
     from employee
     where sal < (select sal
                  from employee
                  where ename = 'FORD');
```

2. Multiple-Row SubQuery: In a subquery if inner query returns more than one row or value as a result to be used as a comparision value for outer query then such queries are called multiple row subquery.

The following are multiple row subquery comparision operator.

a. IN Operator
b. ANY Operator
c. ALL Operator

@ 'IN' Operator: If outer query has to check the existence of a value with a list of values returned by inner query then 'IN' operator is used.

→ WAQ to display the name of an employee who are getting highest salary within each department.

```
SQL> select ename, deptno, sal
     from employee
     where sal IN (select max(sal)
                   from employee
                   group by deptno);
```

(b) 'ANY' Operator: Any operator is used to check a list of any value returned by inner query. Here '>ANY' means greater than maximum value in the list. and '<ANY' means less than minimum value in the list.

→ WAQ to display names of employees who are getting the salary more than the salary of any employee in deptno '20'.

```
SQL> select ename, deptno, sal
     from employee
     where sal > ANY (select sal
                      from employee
                      where deptno = '20');
```

(c) 'ALL' Operator: All operator works similar to 'any' operator and used as '>ALL' means greater than maximum value in the list and '<ALL' means less than minimum value in the list.

Subquery in multiple columns:

We can specify multiple columns also in subquery or 'condition' field.

→ WAQ to display names of employees who are joined in department on joining year of any employee working in deptno '10' and his salary.

```
SQL> select ename, deptno, hiredate, sal
     from employee
     where (to_char(hired date, 'yyyy'), sal) in
           (select to_char(hired date, 'yyyy'), sal
            from employee
            where deptno = 10);
```

Co-related Subquery: In a subquery if inner query returns a value is changed for each row retrieved by outer query then such queries are called co-related subquery.

In these queries the value on the column of a row retrieved by outer query will be supplied as

a comparision value for an inner query.

→ WAQ to display details of employees who are earning salary more than avgsal of their corresponding department.

```
SQL> select empno, ename, deptno, sal
     from employee e1
     where sal > (select Avg(sal)
                  from employee e2
                  where e2.deptno = e1.deptno);
```

→ WAQ to display top 3 earners from all the employee.

```
SQL> select empno, ename, deptno, sal
     from employee e1
     where 3 > (select count(*)
                from employee e2
                where e2.sal >= e1.sal);
```

Subquery within the 'having' clause:-

We can specify the subquery within the 'having' clause also.

→ WAQ to display the name of a dept containing highest number of employees. along with count

```
SQL> select d.name, count(e.empno)
     from employee e, dept d
     where e.deptno = d.deptno
     having count(e.empno) = (select max(count(*))
                              from employee
                              group by deptno);
```

EXIST Operator: Exist operator is used frequently in the subqueries to check the existence of the values retuen by the inner query. If inner query returns any row then exist is a evaluated to 'true' then row of outer query will be displayed into the result. If inner query doesnot produce then exist operator is evaluated to false and row of the outer query will be ignored to display.

→ WAQ to display details of the employees to whom atleast one employee is reporting to them.

```
SQL> select *
     from employee e1
     where exist (select * from employee e2
                  where e2.empno = e1.mgr);
```

Views (Virtual Tables) in SQL:

Views in sql are virtual tables contains deri information by consulting one or more database tab which are called as base tables. Views donot contain /stores information physically as stored in basetables.

The following is the syntax to create a view:

```
create view <view_name>
as select */[columns/aggregate funcs
    from table(s)
    where <condition(s)>
    group by <column(s)>
    having <condition(s)>
```

Once the views are created, the end user directly can access information from the view or the application developer can include these implemented views into the application programs through which end user will have in contact with database system.

The following view V_1 contains the info associated with each employee along with department name.

```
create view V1
as select e.empno, e.ename, e.sal, e.deptno,
        d.deptname
    from employee e, department d
```

where e.deptno = d.deptno; ↵

view is created

Ø Upon creation of view we can retrieve the data through select statement referring the view name as

> select * from v1;

The following view v2 contains the name of the department along with no.of employees.

```
create view v2
as select d.dname, count(e.empno)
   from employee e, department d
   where e.deptno = d.deptno
   group by d.dname;
```

The created views can also be dropped permanently from db

> drop view <view_name>;

Whenever any modification occurs with base tables by insulting new rows, deleting rows or updating any rows those changes will automatically effect / carry on those views which references theses tables as base tables.

Views can be implemented by using two ways:

1. Query modification approach: In this method whenever retrieval requests are specified on created views each time the associated sql query will be

executed by dbms software

2) View materialization Approach: In this method whenever user places retrieval request to the created view, the associated sql statement with that view is executed only once and stores that information temporarily, in future retrievals to the view will refer from this temporary table. But in this method if any updations occurs in basetables those are not automatically carried with view, explicitly the user has to handle this situation.

→ View Created views are allowed for updations also but not recommendable.

→ Schema change statements in sql:

1. Alter
2. drop
3. Truncate
4. Rename