Instagram:- @purohit_17

Parth Purohit

S. No.	Date	Title
		SQL.
		RDBMS.
		Data integrity.
		SQL statements.
		SQL * Plus statements.
		OPerators.
		Order by clause.
		Function.
		Group by clause.
		Sub-queries.
		Joins.
		Co-related subqueries.
		DDL.
		DML.
		TCL.
		DCL
		Normalization.
		Anamolies.
		Views.

SQL ⇒

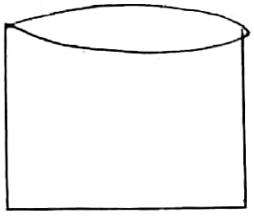
→ Structured Query language.

DATA ⇒

→ raw material
→ Basically data is a raw material, when we processed it, it will converted as useful information.
→ Data can also be defined as set or collection of useful information.
→ Data is a fact related to an object.

DATABASE ⇒

→ Database is the systematic collection of data where we can store and retriere the data.



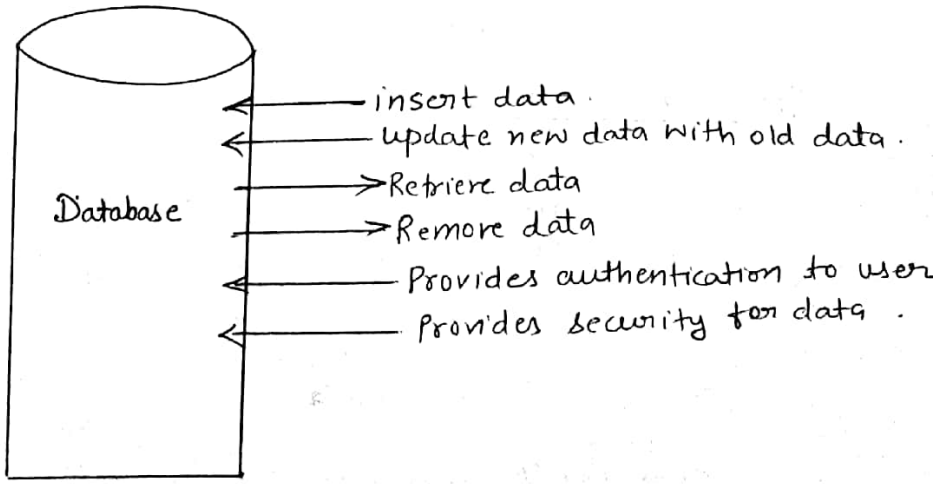
⇒ Representation of database.

DBMS ⇒

→ Database management system.
→ DBMS is the combination of database and management system services.
i.e. DBMS = DB + MSS

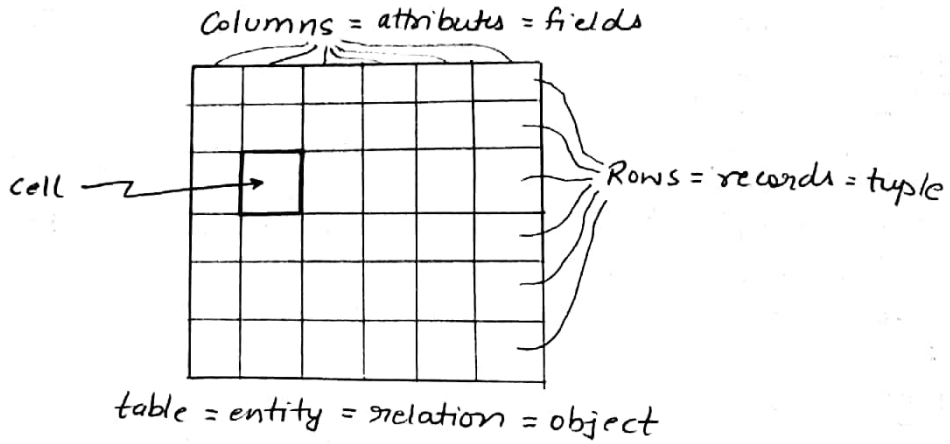
MSS ⇒

→ Management system services access data from the user
→ MSS allows user to access the data.



RDBMS ⇒

→ Relational Database management system.



→ RDBMS defines relationship in the form of table.
→ Table consists of rows and columns.
→ Rows are also called as records or tuple.
→ Columns are also called as attributes or fields.
→ The intersection of rows and columns is called as cell.
→ Table is also called as entity, relation or object.
→ All the latest database products use RDBMS module.

Data integrity ⇒

→ Data integrity is used to restricts invalid data into the table.
→ We can achieve data-integrity by
  (i). Data type.
  (ii) Constraints.

(i). Data-type ⇒

→ Data-type is a type of data which we store in each columns of the table.
→ Here we have three types of datatype
  (a). Number
  (b). Char / Varchar.
  (c). Date

(a). Number ⇒

→ Number is a datatype where we are inputing numeric data or values.

Eg.

(i). Number (5)
-99999 to 99999

(ii). Number (4)
-9999 to 9999

(iii). Number (5,3)
-99.999 to 99.999

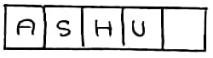
(iv). Number (9,5)
-9999.99999 to 9999.99999

(b). Char / Varchar ⇒

→ Char/Varchar datatype are used to store alpha numeric character.

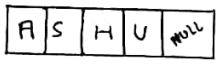
NOTE:

Char(5):



max. of 2000 characters.

Varchar(5):



max of 4000 character

(c). Date ⇒

→ We use Date datatype to enter only valid date.
i.e. dd/mm/yyyy

(ii). Constraints ⇒

→ Constraints are the condition which restricts the invalid data into the table and it is provided to the columns of the table.

Types of constraints ⇒

(a) NOT NULL
(b). UNIQUE
(c). Primary key.
(d). Foreign key.
(e). check.

(a). NOT NULL ⇒

NULL ⇒

→ NULL is nothing.
→ NULL is neither a zero nor a blank space.
→ NULL never occupy space in the memory.
→ If we perform any arithmetic operations with null, results in NULL itself.
→ Two nulls are never same in Oracle.
→ NULL represents unknown value.

NOT NULL ⇒

→ NOT NULL is a constraint where it will ensure atleast some value should be present in a column.

(b). **UNIQUE** ⇒

→ UNIQUE is a constraint where it will not accept duplicate value and it can accept multiple null values

(c). **Primary key** ⇒

→ Primary key is the combination of NOT NULL and UNIQUE constraints.

→ Primary key is a column which uniquely identifies a row.

→ Only one primary key is allowed in a table.

→ Creation of primary key is not mandatory but it is highly recommended.

Emp No.	Emp name	Salary	Hire date
01.	ASHU	1,00,000	20-07-2017
02.	BASU	35,000	20-05-2016
03.	CHHOTI	25,000	14-04-2015
04.	SHALU	25,000	12-02-2012
05.	MADHU	25,000	20-08-2017
06.	AMIT	30,000	17-04-2016

Primary Key

Table : Employ table

Candidate Key ⇒

Columns which are eligible to become primary key, are called as candidate key.

Emp No.	Emp name	Salary	Hire date	Emp Id
01.	ASHU	1,00,000	20-07-2017	A@gmail.com
02.	BASU	35,000	20-05-2016	B@gmail.com
03.	CHHOTI	25,000	14-04-2015	C@gmail.com
04.	SHALV	25,000	12-02-2012	S@gmail.com
05.	MADHV	25,000	20-08-2017	M@gmail.com
06.	AMIT	30,000	17-04-2016	Am@gmail.com

(c). Alternate key ⇒

→ Columns which are eligible to become primary key but not choosen as primary key.

→ Eg. Email Id

Candidate Key	=	Primary Key	+	Alternate key.

(d). Check ⇒

→ Check is a constraint which is used to provide additional validation as per the customer requirement specification.

Eg. Check (salary > 0)

(e). Foreign key ⇒

Emp. Id	Emp name	Salary	Dept no.	D. name	Location
01	ASHU	1,00,000	30	Testing	B'lore
02	BASU	35,000	20	HR	Pune
03	CHHOTI	25,000	10	Developing	Mysore
04	SHALU	25,000	10	Developing	Mysore
05.	MADHU	25,000	20	HR	Pune

Table. Employ table

Foreign key

Emp. Id	Emp. name	Salary	Dept. no.
01	ASHU	1,00,000	30
02.	BASU	35,000	20
03.	CHHOTI	25,000	10
04.	SHALU	25,000	10
05.	MADHU	25,000	20

Child table

Table: Employ table

Primary key

Dept no.	D name	Location
10	Developing	Mysore
20	HR	Pune
30	Testing	B'lore

Master table

Table: Dept table

→ Foreign key constraint is also called as referential integrity constraint.

→ Foreign key creates relationship between two tables.

→ Foreign key is created on the child table.

→ Foreign key can take both null and duplicate value.

→ To create foreign key the master table should have primary key defined on the common column of the master table.

→ We can have more than one foreign key in a table.

Emp. table

Emp Id	Emp. name	Salary	Dep. No.	PID
01	ASHU	1,00,000	30	011
02	BASU	35,000	20	012
03	CHHOTI	25,000	10	012
04	SHALU	25,000	10	011
05	MADHU	25,000	20	011

Child table →

Project table

PID	Pname
011	abcd
012	efgh

Dept. table

Dep. N.	Dname	location
10	Developing	Mysore
20	HR	Punc
30	Testing	B'lore

Master table

* SQL statements ⇒

DQL ⇒ Data query language.
DDL ⇒ Data definition language.
DML ⇒ Data manipulation language.
TCL ⇒ Transaction control language.
DCL ⇒ Data control language

DQL ⇒

SELECT

→ 'Select' command is used to retrieve the data from the database.

DDL ⇒

→ Create. : used to create a new table.
→ Alter : Modify the structure of the table.
→ Rename. : used to rename the table name.
→ Truncate:
→ Drop.

DML ⇒

→ Insert : used to insert records into the table.
→ Update : used to modify the records data.
→ Delete : used to delete the record's data

TCL ⇒

→ Commit : used to save DML changes.
→ Rollback : To restore earlier values
→ Savepoint : used to save the statement names

DCL ⇒

→ Grant : used to give permission access or privileges to user.
→ Revoke :

Commands ⇒

(1). Show user	Command to display the current user
(2). Clear screen or cl scr	clear the screen
(3). Exit	Exit the opp.
(4). Select * from TAB;	Query used to display list of tables which are present in current user.
(5). Describe EMP or Desc EMP;	
(6). Show linesize; Show Pagesize;	Command to show the linesize and pagesize.
(7). SET linesize 120; SET Pagsize 130;	Command to set the linesize and pagesize with new values.
(8). SELECT * FROM EMP;	

Capabilities of SELECT statements ⇒

(1). Projection / Selection of columns.
(2). Selection / Selection of rows.
(3). Joins / Selection of information from multiple tasks.

(1). Projection ⇒

→ It is used to display the selected column.

Syntax:

```
SELECT  */{Distinct}   Expression/column.name [Alias...]
FROM <table_name>.
```

Distinct:

→ It will used to display only the unique values from the column.

Expression:

→ It is used to perform some calculations or operations.

eg. SELECT 1+2
FROM EMP;

DUAL ⇒

→ DUAL is a dummy table used for independent calculation or operation

eg.
```
SELECT 1+2
FROM DUAL;
```

Alias ⇒

→ Alias is the alternative name given to the column heading.

→ Alias name should mention next to the column name.

→ Between the column name and alias name we can use 'as' keyword.
Using 'as' keyword is not mandatory.

→ Alias should be enclosed by double quotes if it is having special character.

→ Underscore can be considered as character in alias.

eg.
```
SELECT ENAME EMPLOYEE NAME          X
FROM EMP;

SELECT ENAME EMPLOYEE_NAME          ✓
FROM EMP;

SELECT ENAME "EMPLOYEE NAME"        ✓
FROM EMP;
```

Literals ⇒

→ Literals can be a number or character or data. Literals are case sensitive.

i.e. Literals → number, character, date.

→ If the literals are number, then no need of enclosing that within single quotes.

→ If the literals are character or date, then it should be enclosed by single quote.

Concatenation operator ⇒

→ Concatenation operator is used to join two string.

→ It is represented by two vertical bars (||).

EDIT command ⇒

→ Edit command is used to modify the previous query.

→ ED can also be used in place of EDIT.

Q. Write a query to display employee name, salary, annual salary of all the employees.

⇒
```
SQL > SELECT ENAME, SAL, SAL*12 AN_SAL
      FROM EMP;
```

Q. Write a query to display ename, salary & annual salary with monthly bonus of 500 rupees.

→ SQL > SELECT ENAME, SAL, (SAL+500)*12 AN_SAL
FROM EMP;

LIST command →

→ LIST is a command used to display previously executed SQL statements.

```
SQL > LIST
      SELECT EMP.*, EMP.ENAME
      FROM EMP;
```

SAVE command →

→ SAVE is a command used to save the previously executed query.

Syntax:

```
SAVE "location"
```

(.txt .doc)

eg. SAVE "D:\\P1.txt";

REPLACE Command →

→ Replace is a command which is used to overwrite the existing document.

Syntax:

```
SAVE "location" REPLACE
```

eg. SAVE "D:\\P1.txt" REPLACE;

APPEND command →

Syntax:

```
SAVE "location" APPEND
```

eg. SAVE "D:\\P2.txt" APPEND;

Spooling →

→ To save statements along with the output, we use spool command.

```
SQL > SPOOL "D:\\P3.txt"
    > SELECT * FROM EMP;
    > SELECT ENAME FROM DEPT;
    > SELECT ENAME FROM EMP;
    :

SQL > SPOOL OFF
```

SQL * Plus ⇒

→ SQL * Plus is a client tool installed along with database.
→ It has specific set of commands.
→ It acts as an interface between user & database.

Difference between SQL and SQL * Plus ⇒

SQL	SQL * Plus.
→ SQL is a standard language to communicate with database.	→ It is an environment which support to write and execute query.
→ SQL is ANSIC standard.	→ It belongs to oracle proprietory (tool developed by oracle).
→ In SQL, semicolon (;) is mandatory	→ In SQL * Plus, semicolon is not mandatory.
→ SQL commands cannot be abbreviated. eg. SELECT, CREATE, UPDATE, DELETE, etc.	→ SQL * Plus commands can be abbreviated eg. clear screen (cl scr) Edit (Ed) List (l) etc.
→ SQL commands can manipulate tables and data definitions.	→ We cannot manipulate tables and data definitions.

Operators ⇒

→ Operators are the symbols used to perform some specific operations.

Operands ⇒

→ Operands are the inputs required for the operations.

Types of operators ⇒

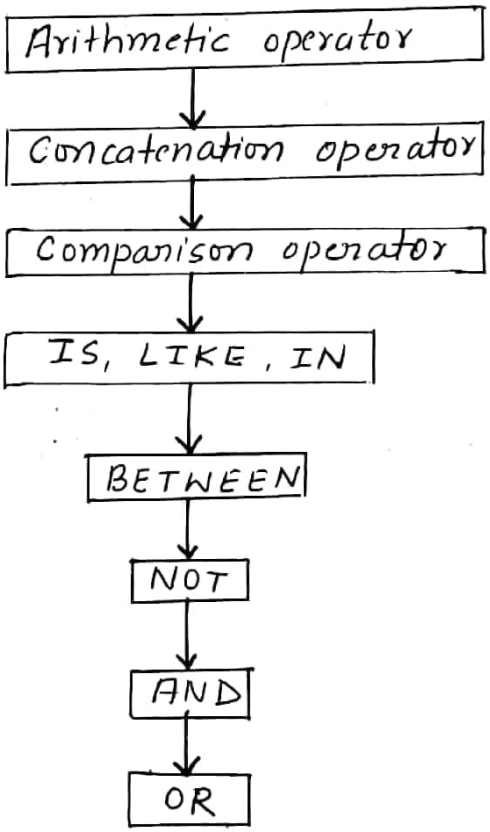
(1). Arithmetic operator. (+, -, *, / ...)

(2). Comparison or Relational operator. (>, <, >=, <=, ...)

(3). Logical operator. (AND, OR, NOT)

(4). Special operator. (IS, LIKE, IN, BETWEEN).

Precedence of operators ⇒




```
SELECT */{[Distinct] Expression/column.name [Alias]}
FROM <table_name>
WHERE <condition/s>;
```

Q.1. Write a query to display list of all employees whose salary is greater than 1000.

⇒
```
SELECT *
FROM EMP
WHERE SAL >1000;
```

Q.2. Write a query to list all the details of employees who are working as manager.

⇒
```
SELECT *
FROM EMP
WHERE JOB = 'MANAGER';
```

Q.3. Write a query to display all the employees who are working in department 20 and working as salesman.

⇒
```
SELECT *
FROM EMP
WHERE DEPTNO =20 AND JOB = 'SALESMAN';
```

Q.4. Write a query to display the list of employees who is working as a manager and salary >2000 and who works with department 30.

⇒
```
SELECT *
FROM EMP
WHERE JOB = 'MANAGER' AND SAL >2000 AND DEPTNO = 30;
```

Q.5. Write a query to display the salary of scott.

⇒
```
SELECT SAL
FROM EMP
WHERE ENAME = 'SCOTT';
```

Q.6. Write a query to display the list of employee who are working in dept 10, 20 and 30.

⇒
```
SELECT *
FROM EMP
WHERE DEPTNO = 10 OR DEPTNO = 20 OR DEPTNO = 30;
```

Q.7. Write a query to display the list of employees who are working as clerk, salesman, manager and analyst.

⇒
```
SELECT *
FROM EMP
WHERE JOB = 'CLERK' OR JOB = 'SALESMAN' OR
      JOB = 'MANAGER' OR JOB = 'ANALYST';
```

IN ⇒

→ Instead of using multiple OR, we are using IN operator.

→ Syntax:

```
SELECT <column_list>
FROM <table_name>
WHERE column_name IN (V1, V2, V3, ... Vn);
```

Q.1. Write a query to display the list of employees who are working as clerk, salesman, manager, analyst.

⇒
```
SELECT *
FROM EMP
WHERE JOB IN('CLERK', 'SALESMAN', 'MANAGER', 'ANALYST')
```

Q.2. Write a query to list all the employees who are working in dept 10 and 20 and earning more than 500.

⇒
```
SELECT *
FROM EMP
WHERE DEPTNO IN (10,20) AND SAL > 500;
```

Q.3. Write a query to display the list of employee who are working as clerk & salesman and department number 20 and 30 where salary should be greater than 1500.

⇒
```
SELECT *
FROM EMP
WHERE JOB IN('CLERK', 'SALESMAN') AND
      DEPTNO IN(20,30) AND SAL >1500;
```

Q.4. Write a query to display the list of employees for not working in department 20 & 30.

⇒
```
SELECT *
FROM EMP
WHERE DEPTNO NOT IN(20,30);
```

BETWEEN ⇒

→ If we want to evaluate the values, we go for BETWEEN operator.

→ Syntax:

```
SELECT <column_list>
 FROM <table_name>
WHERE column_name BETWEEN value1 AND value2;
```

Q.1 Write a query to list all the employees whose salary is between 1000 to 3000.

⇒
```
SELECT *
FROM EMP
WHERE SAL BETWEEN 1000 AND 3000;
```

Q.2. Write a query to display employee name, job & salary where salary ranges between 1000 to 3500 and working as salesman and analyst.

⇒
```
SELECT ENAME, JOB, SAL
FROM EMP
WHERE SAL BETWEEN 1000 AND 3500
        AND JOB IN ('SALESMAN', 'ANALYST');
```

Q.3. Write a query to display employee name and reporting manager employee ID where the employees having reporting manager ID as 7566.

⇒
```
SELECT ENAME, MGR
FROM EMP
WHERE MGR = 7566;
```

LIKE ⇒

→ To match the pattern.

Syntax:

```
SELECT *
FROM <table_name>
WHERE column_name / Expression LIKE 'PATTERN'
                                [Escape 'escape_character']
```

→ In pattern matching, inside patterns we use wildcards or special characters.

(i). Percentage (%) ⇒

→ It matches with n characters

(ii). Underscore (_) ⇒

→ It matches with a single character

→ Like operator is used for pattern matching.

→ Pattern always enclosed by single quote.

Q.1. Write a query to display the employee name start with S.

⇒
```
Select ENAME
From EMP
Where ENAME LIKE 'S%';
```

Q.2. Write a query to display the employee name having letter L as second character.

⇒
```
SELECT ENAME
FROM EMP
WHERE ENAME LIKE '_L%';
```

Q.3. Write a query to display employee name whose name consist of atleast two L

⇒
```
SELECT ENAME
FROM EMP
WHERE ENAME LIKE '%L%L%';
```

Q.4. Write a query to display the employee name whose employee name have R in their third character.

⇒
```
SELECT ENAME
FROM EMP
WHERE ENAME LIKE '__R%';
```

Q.5. Write a query to display all the employee details in which employee name consists of two consecutive L.

⇒
```
SELECT ENAME
FROM EMP
WHERE ENAME LIKE '%LL%';

SELECT *
FROM EMP
WHERE ENAME LIKE '%LL%';
```

Q.6. Write a query to display the list of employees in which the employee name consist of underscore in that.

⇒
```
UPDATE EMP
SET ENAME = 'S_MITH'
WHERE ENAME = 'SMITH';

SELECT *
FROM EMP
WHERE ENAME LIKE '%?_%' ESCAPE '?';
```

IS ⇒

→ IS operator is used to compare NULL values.

→ Syntax:

```
SELECT *
FROM <table_name>
WHERE Column_name IS NULL;
```

Q.1. List all the employees who don't have commission.

⇒
```
SELECT *
FROM EMP
WHERE COMM IS NULL;
```

Q.2. Write a query to list all the employees who have commission.

⇒
```
SELECT *
FROM EMP
WHERE COMM IS NOT NULL;
```

Q.3. Write a query to display all the employees where employee name doesn't start with S

⇒
```
SELECT *
FROM EMP
WHERE ENAME NOT LIKE 'S%';
```

Q.4. Write a query to display all the employees who are having atleast two A's and salary in the range of 1200 to 2800 in department 10, 20 & 30.

⇒
```
SELECT *
FROM EMP
WHERE JOB LIKE '%A%A%'
AND SAL BETWEEN 1200 AND 2800
AND DEPTNO IN (10,20,30);
```

Q.5. Write a query to display all the employees who are not having any reporting manager with number ending with 8 and working as manager or salesman in department 30.

⇒
```
SELECT * FROM EMP
WHERE MGR NOT LIKE '%8' AND DEPTNO IN(30)
AND JOB IN('SALESMAN','MANAGER');
```

Q.6. Write a query to display atleast five characters in their employee name.

⇒
```
SELECT *
FROM EMP
WHERE ENAME '_____%';
```

Q.7. Write a query to display only five character in their employee name.

⇒
```
SELECT *
FROM EMP
WHERE ENAME '_____';
```

Q.8. Write a query to list all the employees who are working as manager or salesman in department 20 and 30 with salary in the range of 1000 to 3000 and having commision.

⇒
```
SELECT *
FROM EMP
WHERE JOB IN('MANAGER','SALESMAN') AND
      DEPTNO IN(20,30) AND
      SAL BETWEEN 1000 AND 3000 AND
      COMM IS NOT NULL;
```

Q.9. Write a query to list all the employees whose salary is in the range of 1000 to 2000 in dept 10, 20, 30 except all clerks.

⇒
```
SELECT *
FROM EMP
WHERE JOB NOT IN('CLERK') AND
 DEPTNO IN(10,20,30] AND
 SAL BETWEEN 1000 AND 2000;
```

Order by ⇒

→ Syntax :

```
SELECT *
FROM <table_name>
WHERE <condition/s>
ORDER BY <column_list>

                <column_list> asc ;
                <column_list> desc ;
```

→ To sort the record in ascending or descending order, we are using ORDER BY clause.

→ It should be the last statement SQL

→ If we want to arrange or sort the records in ascending order, we are using asc keyword

→ If we want to arrange or sort in descending order, we are using desc keyword.

→ If we wont specify asc or desc keyword, by default it will be sorted in ascending order.

Q.1. Write a query to display the deptno in ascending order.

⇒
```
SELECT *
FROM EMP
ORDER BY DEPTNO ;
```

Q.2. Write a query to display the deptno. in descending order.

⇒
```
SELECT *
FROM EMP
ORDER BY DEPTNO DESC;
```

Q.3. Write a query to list all the employee who are working as salesman and sort their salary in ascending order.

⇒
```
SELECT *
FROM EMP
WHERE JOB = 'SALESMAN'
ORDER BY SAL ;
```

Examples ⇒

Q.1. Display all the employees whose name starts with M.

⇒
```
SELECT *
FROM EMP
WHERE ENAME LIKE 'M%';
```

Q.2. Display all the analyst and clerk in department no. 10 and 20.

⇒
```
SELECT *
FROM EMP
WHERE JOB IN ('CLERK', 'ANALYST') AND
      DEPTNO IN (10, 20);
```

Q.3. List all the department name which are having blank space in their location & having dept. name with atleast two E's.

⇒

Q.4. Display all the employees who are having atleast two A's in their job description and salary in the range 1200 to 2800 in dept 10, 20, 30.

⇒
```
Select *
From EMP
where JOB LIKE '%A%A%' AND
      SAL BETWEEN 1200 AND 2800 AND
      DEPT IN (10, 20, 30);
```

Q.5. Display all the employees who are not having any reporting manager with number ending with 8 and not working as manager or salesman in dept. 30.

⇒

Q.6. List all the employees who are not getting any commission with their designation neither clerk nor manager and getting salary more than 1500.

⇒
```
SELECT *
FROM EMP
WHERE COMM IS NULL AND
      JOB NOT IN ('CLERK', 'MANAGER') AND
      SAL >1500 .
```

Q.7. List all the employees whose name is having atleast five characters and having reporting manager with salary in the range 800 to 2000 in the dept 30 or 40.

⇒
```
SELECT *
FROM EMP
WHERE ENAME LIKE '_____' AND
      MGR IS NOT NULL AND
      SAL BETWEEN 800 AND 2000 AND
      DEPTNO IN (30,40);
```

Q.8. Display all the employees whose name starts with S and doesn't end with T.

⇒
```
SELECT *
FROM EMP
WHERE ENAME LIKE 'S%' AND
      ENAME NOT LIKE '%T';
```

Q.9. Display all the employee who is working as manager for only one employee.

⇒

Q.10. List all the employees whose job is salesman and salary ranges between 1500 to 3000.

⇒
```
SELECT *
FROM EMP
WHERE JOB = 'SALESMAN' AND
      SAL BETWEEN 1500 AND 3000;
```

Q.11. List all the employees whose salary is in the range of 1000 & 2000 in dept 10,20,30 except all clerks.

⇒
```
SELECT *
FROM EMP
WHERE SAL BETWEEN 1000 AND 2000 AND
      DEPTNO IN (10,20,30) AND
      JOB != CLERK ;
```

Q.12 List all the employees whose name start with letter A and whose salary is greater than 1000.

⇒
```
SELECT *
FROM EMP
WHERE ENAME LIKE 'A%' AND SAL > 1000 ;
```

Q.13. Display all the employee whose name has consecutive M in it.

⇒
```
SELECT *
FROM EMP
WHERE ENAME LIKE '%MM%' ;
```

Q.14. Display all the employees whose job not belong to clerk and whose deptno. is 10 and 20.

⇒
```
SELECT *
FROM EMP
WHERE JOB != CLERK AND DEPTNO IN (10,20);
```

Q.15. List all the employee that doesnot start with A and having salary more than 1500.

⇒
```
SELECT *
FROM EMP
WHERE ENAME NOT LIKE 'A%' AND SAL >1500;
```

Q.16. List all the employees whose employee number is 7654.

⇒
```
SELECT *
FROM EMP
WHERE EMPNO = 7654;
```

Q.17. Display all the salesman in deptno 30.

⇒
```
SELECT *
FROM EMP
WHERE JOB = 'SALESMAN' AND DEPTNO = 30 ;
```

Q.18. Display all the employees whose job name end with second character E.

⇒
```
SELECT *
FROM EMP
WHERE JOB LIKE '%E_' ;
```

Q.19. Display all the employees who are manager and display their annual salary.

⇒
```
Select ENAME, SAL, SAL*12 ANNSAL
From EMP
Where JOB = 'MANAGER'
```

Q.20. List all the employees with employee number between 7500 to 8000.

⇒
```
SELECT *
FROM EMP
WHERE EMPNO BETWEEN 7500 AND 8000;
```

Q.21. List all the employees with annual salary with 500 Rs bonus and working as manager.

⇒
```
SELECT EMP, SAL, (SAL*12)+500 ANNSAL
FROM EMP
WHERE JOB = 'MANAGER';
```

Q.22. Display employee who are earning salary more than 300 but not more than 3000.

⇒
```
SELECT *
FROM EMP
WHERE SAL BETWEEN 300 AND 3000;
```

Q.23. Display the employee whose second character from beginning is A and the last second letter is R.

⇒
```
Select *
From EMP
Where ENAME LIKE '_A%' AND ENAME LIKE '%R_';
```

Q.24. List the employees, job with hire date whose name starts with A or S & working in department 20 & 30.

⇒
```
SELECT ENAME, JOB, HIREDATE
FROM EMP
WHERE ENAME LIKE 'A%' OR ENAME LIKE 'S%' AND
      DEPTNO IN (20,30);
```

Q.25. Write a query to display departmentname which contain atleast 2S and atleast 1 of O or A in their location.

⇒
```
SELECT DNAME
FROM DEPT
WHERE LOC LIKE '%S%S%' AND LOC LIKE '%O%' OR
      LOC LIKE '%A%';
```

IG:- @purohit_17

Q.26. List all the employee with annual salary with 5000 rupees bonus working as monager, salesman and analyst.

⇒
```
SELECT EMP, SAL, (SAL * 12) + 5000 ANNSAL
FROM EMP
WHERE JOB IN ('MANAGER', 'SALESMAN', 'ANALYST');
```

Q.27. List the employee in department 20 who have salary less than 500.

⇒
```
SELECT *
FROM EMP
WHERE DEPTNO = 20 AND SAL < 500;
```

Q.28. List all the distinct department number from employee table.

⇒
```
SELECT DISTINCT DEPTNO
FROM EMP;
```

Q.29. List all the salgrade whose low salary between 1000 to 2000.

⇒
```
SELECT ENAME, SALGRADE
FROM EMP, SALGRADE
//WHERE SAL = LOSAL AND SAL BETWEEN
                    1000 AND 2000
```

Q30. List all the employee except who are having commission.

⇒
```
SELECT *
FROM EMP
WHERE COMM IS NOT NULL;
```

Q.31. List all the employee names whose names start with K and J.

⇒
```
SELECT *
FROM EMP
WHERE ENAME LIKE 'K%' OR ENAME LIKE 'J%';
```

Q.32. List all the employee names whose name starts with A and S.

⇒
```
SELECT *
FROM EMP
WHERE ENAME LIKE 'A%' OR ENAME LIKE 'S%';
```

Q.33. List all the employee whose manager employee number between 7698, 7839 and salary ends with number 5

⇒
```
SELECT *
FROM EMP
WHERE MGR BETWEEN 7698 AND 7839 AND
      SAL LIKE '%5';
```

Q.33. List all the employee whose employee third letter should be 2 and is manager with annual salary between 30,000 to 35,000.

⇒
```
SELECT *
FROM EMP
WHERE ENAME LIKE '__2%' AND
    SAL BETWEEN 30,000 AND 35,000 AND
    JOB = 'MANAGER';
```

Q.34. Display the names of employee working in dept 10,20,30 and working as clerk, salesman or analyst.

⇒
```
SELECT ENAME
FROM EMP
WHERE DEPTNO IN(10,20,30) AND
        JOB IN('CLERK','SALESMAN','ANALYST');
```

Q.35. Display the details of the employee whose commission is more than their salary.

⇒
```
SELECT *
FROM EMP
WHERE COMM>SAL;
```

Q.36. List all the analyst in all the department.

⇒
```
SELECT *
FROM EMP
WHERE JOB='ANALYST';
```

Q.37. List all the employee whose commission is greater than 100.

⇒
```
SELECT *
FROM EMP
WHERE COMM>100;
```

Q.38. Display all the employee who are manager having E as the last character in their name but salary having exactly 4 characters.

⇒
```
SELECT *
FROM EMP
WHERE JOB='MANAGER' AND ENAME LIKE '%E' AND
        SAL LIKE '____';
```

Q.39. Display all the employee who are having reporting manager and where job has string 'ANA' unit.

⇒
```
SELECT *
FROM EMP
WHERE
        JOB LIKE '%ANA%' AND MGR IS NOT NULL
```

Q 40. Display all the employees whose name starts with B and ends with E & salary is more than 2200.

⇒
```
SELECT *
FROM EMP
WHERE ENAME LIKE 'B%' AND ENAME LIKE '%E' AND
      SAL > 2200;
```

Function ⇒

→ Functions are reusable programs which returns some output.

→ In order to call the program, we should pass input.
Input is also called as arguments or parameter in functions.

General syntax of function ⇒

→
```
FUNCTION function_name (Arg1, Arg2, .... Arg n)
    {
        Statements ;
    }
```

Types of functions ⇒

(1). Built-in function.
(2). User-defined function.

(1). Built-in function ⇒

→ Built-in functions are nothing but pre-defined function, that means the function are ready, just by calling those functions, we are executing queries.

(2). User-defined function ⇒

→ User-defined function is the extensio of SQL or further study about SQL

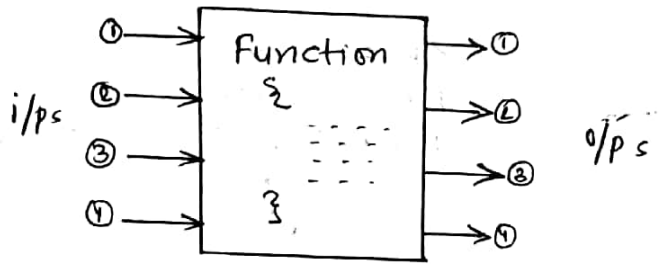
→ Here we come across PL/SQL wh we use control statements, looping, OOPs concept.

Types of built-in function ⇒

(i). Single-row function.
(ii). Multi-row function.

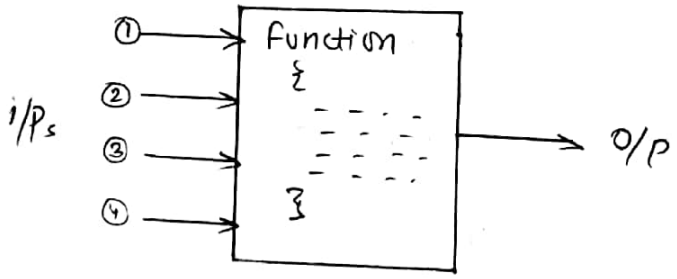
(i). Single-row function ⇒

→ In single row function, we are returning one output for one input.



(ii). Multi-row function ⇒

→ It returns one output for multiple inputs



Types of Single-row function ⇒

(1). Character function.
(2). Number function.
(3). General function.
(4). Date function.
(5). Conversion function.

(1). Character function ⇒

Types:
(a). Case manipulation function.
(b). Character manipulation function.

(a). Case manipulation function ⇒

Types:
(i). Upper function.
(ii) Lower function.
(iii). Init CAP function.

(i). Upper function ⇒

→ Upper function is used to convert letters in a given string to upper case.

(ii). lower function ⇒

→ It is used to convert all the letters in a given string to lower case.

(iii). INIT CAP ⇒

→ It is used to convert first letter of all the word to upper case and rest to lower case in a given function.

```
SELECT *
FROM EMP
WHERE JOB = 'manager';
```

↓

No rows selected.

```
SELECT *
FROM EMP
WHERE JOB = UPPER('manager');
```

```
SELECT *
FROM EMP
WHERE JOB = LOWER('SMITH');
```

```
SELECT *
FROM EMP
WHERE JOB = INITCAP('SMITH');
```

```
UPDATE EMP
SET ENAME = 'smith'
WHERE ENAME = 'CLARK';
```

```
UPDATE EMP
SET ENAME = 'Smith'
WHERE ENAME = 'MILLER';
```

(b). Character manipulation function ⇒

Types:
(i). Concat
(ii). length
(iii). Substr
(iv). Instr
(v). Replace
(vi). Reverse
(vii). Trim.

(i). Concat ⇒
→ It is used to join two strings.
→ Concat function takes only two arguments
→ Syntax:

```
CONCAT (Arg1, Arg2) ;
```

```
SELECT CONCAT ('ENAME','JOB')
FROM DUAL;

SELECT CONCAT ('ENAME ','JOB')
FROM DUAL;

SELECT CONCAT('LET', CONCAT (' ','DANCE'))
FROM DUAL;
```

Q. Write a query whose output should be Let's dance.
⇒
```
SELECT CONCAT ('LET"S', 'DANCE')
FROM DUAL;
```

(ii) length ⇒
→ It is used to find the length of given string.

Q. Write a query to list all the employees whose length of employee name is 5.
⇒
```
SELECT *
FROM EMP
WHERE LENGTH(ENAME) = 5;
```

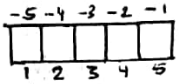
(iii). Substr ⇒
→ It is used to extract substring in the given string.
→ Syntax:

```
SUBSTR (String, Start_position, length)
                                optional
```

Q. List the employee where the third position should be R.
⇒
```
SELECT *
FROM EMP
WHERE SUBSTR (ENAME, 3, 1) = 'R';
```

Q. Write a query to display the employee name where last two character are ER.

⇒
```
SELECT *
FROM EMP
WHERE SUBSTR(ENAME, -2, 2) = 'ES';
```



NOTE:
→ We cannot use 'substr' if we don't know the start position. so for such queries, we go for LIKE operator.

(iv). INSTR ⇒

→ It is used to find the position of the substring in the given string. If substring is not present in the given string, then it returns zero.

→ Syntax:

```
INSTR (String, Substring, String-position, appearance)
                          optional
```

Q. List all the employees whose name starts with S.

⇒
```
SELECT ENAME
FROM EMP
WHERE INSTR(ENAME, 'S', 1, 1) = 1;
```

Q. Write a query to list all employees whose name is having atleast one S in it.

⇒
```
SELECT ENAME
FROM EMP
WHERE INSTR (ENAME, 'S', 1, 1) <> 0;
```

OR

```
SELECT ENAME
FROM EMP
WHERE INSTR(ENAME, 'S', 1, 1) != 0;
```

(v). Replace ⇒

→ Replace function is used to replace sequence of characters in the given string with another set of characters.

→ Syntax:

```
REPLACE (String, String_to_replace,
                 Replacement_string)
```

eg.
```
SELECT REPLACE ('JSPIDERS', 'J', 'Q')
FROM DUAL;

SELECT REPLACE ('JSPIDERS', 'J', 'N')
FROM DUAL;

SELECT REPLACE ('JSPIDERS', 'J', 'S')
FROM DUAL;
```

(vi). Reverse ⇒

→ It is used to reverse the given string.

→ Syntax:

```
REVERSE (string)
```

Q. Write a query to reverse all the employee names in a employee table.

⇒
```
SELECT ENAME, REVERSE(ENAME) R_NAME
FROM EMP;
```

(vii). Trim ⇒

→ It is used to remove some spaces or characters in a given string.

→ Types:

(a). LTRIM
(b). RTRIM
(c). TRIM.

(a). LTRIM ⇒

→ It is used to remove all specified character from left-hand side.

→ Syntax:

```
LTRIM (string, trim_string)
```

eg.
```
SELECT LTRIM ('EEWELCOME', 'E')
FROM DUAL;
```

(b). RTRIM ⇒

→ It is used to remove all specified character from Right-hand side.

→ Syntax:

```
RTRIM (string, trim_string)
```

eg.
```
SELECT RTRIM ('EEWELCOME', 'E')
FROM DUAL;

SELECT RTRIM ('EEWELCOME', 'ER')
FROM DUAL;
```

(c). TRIM ⇒

→ It is used to trim the characters or remove all the specified characters from the given string.

→ Syntax:

```
TRIM ( LEADING/TRAILING/BOTH
        TRIM_CHARACTER
            FROM STRING )
```

eg.

```
SELECT TRIM (LEADING 'E' FROM 'EEWELCOME')
FROM DUAL;
```

o/p ⇒ WELCOME

```
SELECT TRIM(LEADING 'EW' FROM 'EEWELCOME')
FROM DUAL;
```

o/p ⇒ error

NOTE ⇒

→ In TRIM, we cannot use two characters as a trim character i.e. only one character is acceptable in trim character.

```
SELECT TRIM (TRAILING 'E' FROM 'EEWELCOME')
FROM DUAL;
```

o/p ⇒ EEWELCOM

```
SELECT TRIM (BOTH 'E' FROM 'EEWELCOME')
FROM DUAL;
```

o/p ⇒ WELCOM

Assignment ⇒

Q. Write a query to list all the employees whose name is palindrome.

⇒

```
UPDATE EMP
SET ENAME = 'MADAM'
WHERE ENAME = 'SMITH';

SELECT ENAME, REVERSE(ENAME)
FROM EMP
WHERE ENAME = REVERSE(ENAME);
```

(2). Number function ⇒

→ It is used to perform calculation.

→ Types:

(a). ROUND.
(b). TRUNC.
(c). MOD.
(d). POWER.
(e). SQRT
(f). ABS

(a). ROUND ⇒

→ To round-off the nearest value, we use round function.

→ Syntax:

```
ROUND (Number)
```

eg.

```
SELECT ROUND(55.6)
FROM DUAL;
```
56

```
SELECT ROUND(54.4)
FROM DUAL;
```
54

(b). TRUNC ⇒

→ It is used to remove all the decimal values.

eg.

```
SELECT TRUNC(55.66)
FROM DUAL;
```
55

(c). MOD ⇒

Q. Write the query to display all the employee details who are getting even salary.

⇒
```
SELECT ENAME, SAL
FROM EMP
WHERE MOD(SAL,2) = 0;
```

Q. Write the query to display all the employee details who are getting odd salary.

⇒
```
SELECT ENAME, SAL
FROM EMP
WHERE MOD(SAL,2) != 0;
```

(d). POWER ⇒

→ Power function is used to find the power of a number.

eg.
```
SELECT POWER(4,3) POWER
FROM DUAL;
```

(e). SQRT ⇒

→ It is used to find the square root of a number

eg.
```
SELECT SQRT(16) SQRT
FROM DUAL;
```

(f). ABS ⇒

→ It is used to find absolute value

eg.

```
SELECT ABS(-90)
FROM DUAL;

SELECT ABS(-90.33)
FROM DUAL;
```

(3). Date function ⇒

→ To display the system date we use SYSDATE

→ To display the current date, we use CURRENT_DATE.

eg.

```
SELECT SYSDATE
FROM DUAL;

SELECT CURRENT_DATE
FROM DUAL;
```

→ SYSTIMESTAMP ⇒

```
SELECT SYSTIMESTAMP
FROM DUAL;
```

Q. Write a query to display the joining dat of all the employees in employee table.

⇒

```
SELECT ENAME, HIREDATE JOIN_DATE
FROM EMP;
```

(4). Conversion function ⇒

→ Conversion function is used to convert one data type into another datatype.

→ Types :

(a). Implicit conversion function.
(b). Explicit conversion function.

(a). Implicit conversion function ⇒

→ It is used to convert one data-type into another without using conversion function.

(b). Explicit conversion function ⇒

→ It is used to convert one data-type into another using conversion funct

Q. Write a query to display the department number using implicit function.

⇒
```
SELECT *
FROM EMP
WHERE DEPTNO = 30;
```

Q. Write a query to display the department number using explicit function.

⇒
```
SELECT *
FROM EMP
WHERE DEPTNO = TO_NUMBER('30');
```

Elements of Date formats :

YYYY ⇒
→ It will display full year in number.

YEAR ⇒
→ year spelled out.

MM ⇒
→ Two digit value for month.

MONTH ⇒
→ Full name of month.

MON ⇒
→ Three letter abbreviation of month.

DY ⇒
→ Three letter abbreviation of day.

DD ⇒
→ Numeric day of month.

DAY ⇒
→ Full name of day.

```
SELECT SYSDATE, TO_CHAR(SYSDATE, 'YYYY* YEAR MM
       MONTH MON DY DD DAY')
FROM DUAL;
```

⇒ 26-MAR-18 2018 TWENTY EIGHTEEN 03 MARCH MON MONDAY 26.

Q. Write a query to display only joining date.

⇒
```
SELECT ENAME, HIREDATE, TO_CHAR(HIREDATE, 'DAY')
FROM EMP;
```

(5). General function ⇒

Types ⇒

(a). NVL (Null value logic)

(b) NVL2

(a). NVL ⇒

→ If the first argument is NULL, it will return argument 2.

→ If argument 1 is not NULL, it will return argument 1 itself.

→ Syntax :

```
NVL(arg1, arg2)
```

Q. Write a query to display total monthly salary.

⇒
```
SELECT ENAME, COMM, SAL, NVL(COMM+SAL, SAL)
FROM EMP;
```

(b). NVL2 ⇒

→
```
NVL2(arg1, arg2, arg3)
```

→ If argument 1 is NULL, it will return argument 3.

→ If argument 1 is not NULL, it will return argument 2.

Q. Write a query to display total monthly salary using NVL2 logic.

⇒
```
SELECT ENAME, COMM, SAL, NVL2(COMM, SAL+COMM, SAL)
FROM EMP;
```

(ii). Multiple-row function ⇒

→ It takes multiple inputs and returns one output.

→ It operates on set of rows and returns one output per group.

→ It is also called as group function or aggregate function.

Types ⇒

(1). Maximum.

(2). Minimum.

(3). Average.

(4). Count.

(5). Sum.

(1). MAX ⇒

→ Here it returns maximum value in a group.

Q. Write a query to display maximum salary of a employee.

⇒ SELECT MAX(SAL) FROM EMP;

(2). MIN ⇒

→ It returns minimum or least value.

Q. Write a query to display minimum salary of a employee.

⇒ SELECT MIN(SAL) FROM EMP;

(3). AVG ⇒

→ It returns average value.

Q. Write a query to display average value of salary.

⇒ SELECT AVG(SAL) FROM EMP;

(4). COUNT ⇒

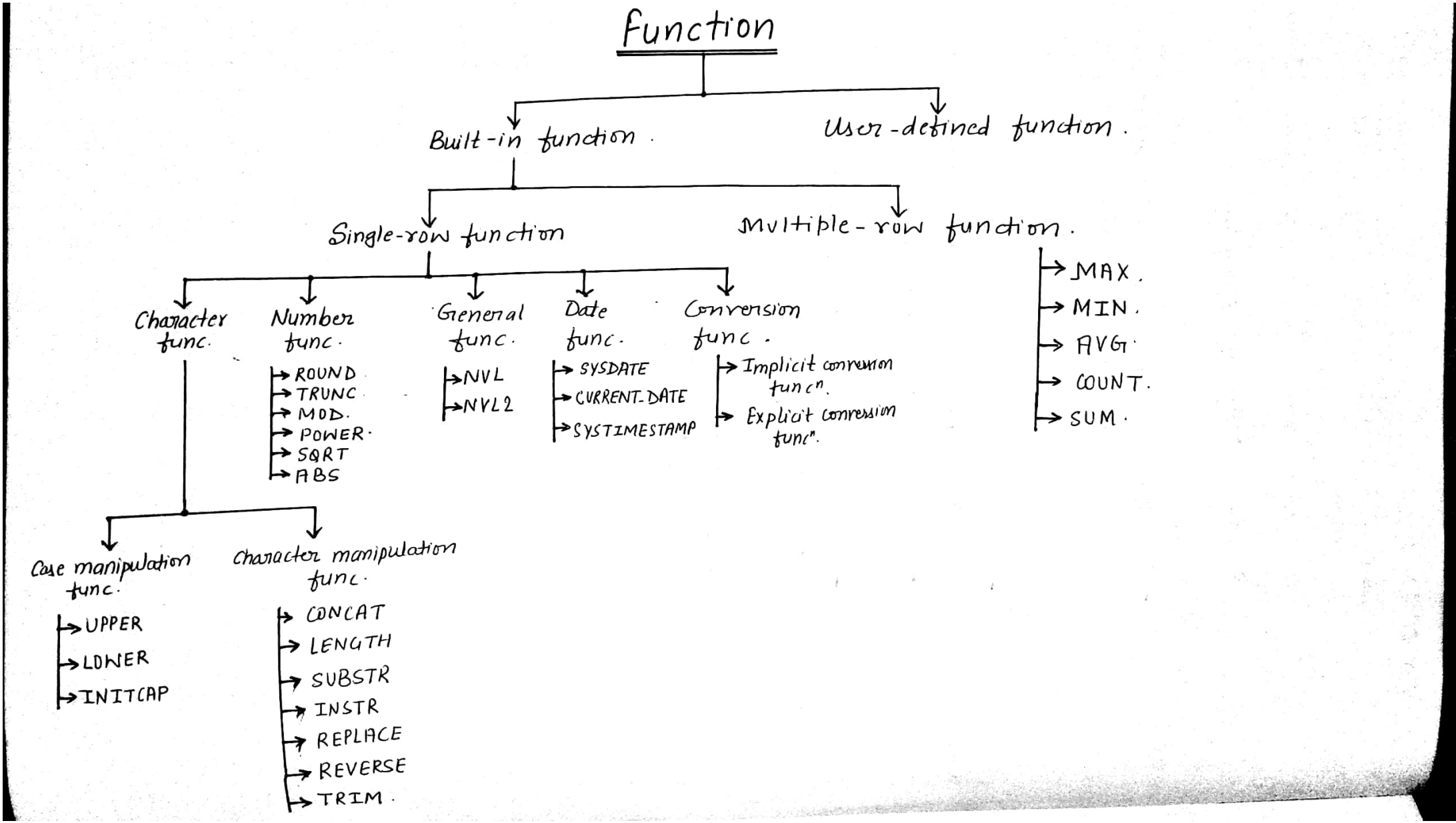
→ It returns total counts of record.

eg. SELECT COUNT(*)
FROM EMP;

(5). SUM ⇒

→ It returns total value.

eg SELECT SUM(SAL)
FROM EMP;



Group by clause ⇒

→ It is used in a SELECT statement to collect data across multiple records and group the result by 1 or more columns.

→ Syntax:

```
SELECT <column_list>, <aggregate functions>
FROM <table_name>
[optional]
WHERE <condition/s>
GROUP BY <column_list>
HAVING <condition_based_on_group_by>
```

Rules ⇒

(1). Whatever the columns are present in SELECT statement, that should be present in group by clause.

(2). Whatever the columns present in group by clause need not to be present in SELECT statement.

(3). WHERE condition is used to check condition before group by clause.

(4). We cannot use aggregate function in WHERE condition.

(5). HAVING clause is used to restrict group data and it is used after group by clause.

(6). Whatever the column present in HAVING clause, those column should be present in group by clause and we can use any aggregate function in HAVING clause.

Q.1. Write a query to display department wise number of employees.

⇒
```
SELECT DEPTNO, COUNT(*)
FROM EMP
GROUP BY DEPTNO
ORDER BY DEPTNO;
```

Q.2. Write a query to display maximum salary, total salary in each of the job.

⇒
```
SELECT JOB, MAX(SAL), SUM(SAL), COUNT(*)
FROM EMP
GROUP BY JOB;
```

Q.3. Write a query to display department wise total salary except all the employees whose name starts with S.

⇒
```
SELECT DEPTNO, SUM(SAL)
FROM EMP
WHERE ENAME NOT LIKE 'S%'
GROUP BY DEPTNO;
```

Q.4. Write a query to display departmentwise average salary except salesman.

⇒
```
SELECT DEPTNO, AVG(SAL)
FROM EMP
WHERE JOB != 'SALESMAN'
GROUP BY DEPTNO;
```

Q.5. Display departmentwise department number which are having more than four employees in it.

⇒
```
SELECT DEPTNO, COUNT(*)
FROM EMP
GROUP BY DEPTNO
HAVING COUNT(*) > 4;
```

Q.6. Write a query to display departmentwise total salary except department number 10.

⇒
```
SELECT DEPTNO, SUM(SAL)
FROM EMP
GROUP BY DEPTNO
HAVING DEPTNO != 10;
```

OR

```
SELECT DEPTNO, SUM(SAL)
FROM EMP
WHERE DEPTNO != 10
GROUP BY DEPTNO;
```

Q.7 Write a query to display jobwise highest salary only if the highest salary is greater than 500.

⇒
```
SELECT JOB, MAX(SAL)
FROM EMP
GROUP BY JOB
HAVING MAX(SAL) > 1500;
```

OR

```
SELECT JOB, MAX(SAL)
FROM EMP
WHERE SAL > 1500
GROUP BY JOB;
```

Q.8. Write a query to display jobwise highest salary only if the highest salary is more than 1500. excluding dept 30 and sort the data based on highest salary in the ascending order.

⇒
```
SELECT JOB, MAX(SAL)
FROM EMP
WHERE DEPTNO != 30
GROUP BY JOB
HAVING MAX(SAL) >1500
ORDER BY MAX(SAL);
```

Q.9. Write a query to display deptno. along with the number of employees in it.

⇒
```
SELECT DEPTNO, COUNT(*)
FROM EMP
GROUP BY DEPTNO;
```

Q.10. Write a query to display the deptno. which are having more than 5 employees in it.

⇒
```
SELECT DEPTNO, COUNT(*)
FROM EMP
GROUP BY DEPTNO
HAVING COUNT(*) >5;
```

Sub-queries ⇒

→ Query within a query is called as sub-query.

→ The innermost query will execute first.

→ We can write maximum of 255 subqueries and it is called as nested subqueries.

→ Subqueries are separated by paranthesis.

Syntax:

```
SELECT - - - .
FROM - - .
WHERE . . . . ( SELECT - - -
                FROM - - .
                WHERE . . (SELECT - - - .
                           FROM - - - .
                           WHERE - - - . (SELECT - - -
                                          FROM - - -
                                          WHERE - - - . ) ) ) ;
```

Types ⇒

(1). Single-row subquery.
(2). Multi-row subquery.

(1). **Single-row subquery ⇒**

→ It is used to return one or single output.

(2). **Multi-row subquery ⇒**

→ It is used to return more than one output.

Q1. Write a query to display the list of all the employees who are earning more than SMITH

⇒
```
SELECT ENAME, DEPTNO, SAL
FROM EMP
WHERE SAL > (SELECT SAL
             FROM EMP
             WHERE ENAME = 'SMITH');
```

Q.2. Write a query to display the list of all employees who are working in the same department of Allen

⇒
```
SELECT ENAME, DEPTNO
FROM EMP
WHERE DEPTNo = (SELECT DEPTNO
                FROM EMP
                WHERE ENAME = 'ALLEN');
```

Q.3. Write a query to list all the employees whose job is same as that of JONES.

⇒
```
SELECT *
FROM EMP
WHERE JOB = (SELECT JOB
             FROM EMP
             WHERE ENAME = 'JONES');
```

Q.4. Write a query to list all the employees who are working in the department of allen and earning more than SMITH.

⇒
```
SELECT *
FROM EMP
WHERE DEPTNO = (SELECT DEPTNO
                FROM EMP
                WHERE ENAME = 'ALLEN') AND
                      SAL > (SELECT SAL
                             FROM EMP
                             WHERE ENAME = 'SMITH');
```

Q.5. Write a query to display the department name of king.

⇒
```
SELECT DNAME
FROM DEPT
WHERE DEPTNO=(SELECT DEPTNO
              FROM EMP
              WHERE FRAME = 'KING');
```

Q.6. Write a query to display the LOC (Location) of 'ALLEN'.

⇒
```
SELECT LOC
FROM DEPT
WHERE DEPTNO = (SELECT DEPTNO
                FROM EMP
                WHERE ENAME = 'ALLEN');
```

Q.7. Write a query to display all the employees who works for Research Department.

⇒
```
SELECT ENAME
FROM EMP
WHERE DEPTNO = (SELECT DEPTNO
                FROM DEPT
                WHERE DNAME = 'RESEARCH');
```

Q.8. Write a query to list all the employee who located at DALLOS.

⇒
```
SELECT ENAME
FROM EMP
WHERE DEPTNO = (SELECT DEPTNO
                FROM DEPT
                WHERE LOC = 'DALLAS');
```

Q.9. Write a query to list all the employees who are working in ACCOUNTING and RESEARCH department.

⇒
```
SELECT ENAME
FROM EMP
WHERE DEPTNO IN (SELECT DEPTNO
                 FROM DEPT
                 WHERE DNAME IN ('ACCOUNTING',
                                 'RESEARCH');
```

Q.10. List the department name that are having analyst.

⇒
```
SELECT ENAME
FROM EMP
WHERE DEPTNO IN ( SELECT DEPTNO
                  FROM EMP
                  WHERE JOB = 'ANALYST');
```

Q.11. Write a query to list all the employees who has salary greater than ALLEN.

⇒
```
SELECT ENAME
FROM EMP
WHERE SAL > (SELECT SAL
             FROM EMP
             WHERE ENAME = 'ALLEN');
```

Q.12. Write a query to list all department names which are having 'SALESMAN' in it.

⇒
```
SELECT DNAME
FROM DEPT
WHERE DEPTNO IN ( SELECT DEPTNO
                  FROM EMP
                  WHERE JOB = 'SALESMAN');
```

Q.13. Write a query to display the employees whose location is having atleast '0' in it.

⇒
```
SELECT ENAME
FROM EMP
WHERE DEPTNO IN( SELECT DEPTNO
                 FROM DEPT
                 WHERE LOC LIKE '%0%');
```

Q.14. Write a query to list the department name that are having atleast 4 emp in it.

⇒
```
SELECT DNAME
FROM DEPT
WHERE DEPTNO IN (SELECT DEPTNO
                 FROM EMP
                 GROUP BY DEPTNO
                 HAVING COUNT(ENAME)> 3);
```

Q.15. Write a query to display deptname which are having atleast 2 clerks in it.

⇒
```
SELECT DNAME
FROM DEPT
WHERE DEPTNO IN ( SELECT DEPTNO
                  FROM EMP
                  WHERE JOB IN ( SELECT JOB
                                 FROM EMP
                                 WHERE JOB = 'CLERK'
                                 GROUP BY JOB
                                 HAVING COUNT(JOB) >1));
```

Q.16. Write a query to display second highest salary.

⇒
```
SELECT MAX(SAL)
FROM EMP
WHERE SAL < (SELECT MAX(SAL)
             FROM EMP);
```

Q.17. Write a query to display all the department names that are having no employees in it.

⇒
```
SELECT DNAME
FROM DEPT
WHERE DEPTNO NOT IN (SELECT DEPTNO
                     FROM EMP);
```

Q.18. Write a query to list all the employees whose JOB is same as SCOTT and their salary is greater than SMITH salary.

⇒
```
SELECT ENAME
FROM EMP
WHERE JOB IN(SELECT JOB
             FROM EMP
             WHERE ENAME ='SCOTT') AND
      SAL > (SELECT SAL
             FROM EMP
             WHERE ENAME = 'SMITH');
```

Q.19. Write a query to display SCOTT manager's manages department name.

⇒
```
SELECT DNAME
FROM DEPT
WHERE DEPTNO (SELECT DEPTNO
              FROM EMP
              WHERE EMPNO = (SELECT MGR
                             FROM EMP
                             WHERE EMPNO = (SELECT MGR
                                            FROM EMP
                                            WHERE ENAME
                                            ='SCOTT');
```

Q.20. Write a query to display all the employees who are actual manager.

⇒
```
SELECT ENAME
FROM EMP
WHERE EMPNO IN (SELECT MGR
                FROM EMP);
```

Q.21. Write a query to select employee no., job & salary for all the analyst who are earning more than any of the manager.

⇒
```
SELECT EMPNO, JOB, SAL
FROM EMP
WHERE JOB='ANALYST' AND
      SAL > (SELECT MIN(SAL)
             FROM EMP
             WHERE JOB = 'MANAGER');
```

Q.22. Write a query to display the dept name and location of all the employees working for CLARK.

⇒
```
SELECT DNAME, LOC
FROM DEPT
WHERE DEPTNO (SELECT DEPTNO
              FROM EMP
              WHERE ENAME = 'CLARK');
```

Q.23. Write a query to display all the employee whose salary is greater than any salary of department 20.

⇒
```
SELECT ENAME, SAL
FROM EMP
WHERE SAL > (SELECT MIN(SAL)
             FROM EMP
             WHERE DEPTNO = 20);
```

Q.24. Write a query to display first employee record based on hire date.

⇒
```
SELECT *
FROM EMP
WHERE HIREDATE = (SELECT MIN(HIREDATE)
                  FROM EMP);
```

Q.25. Write a query to list the department names that are having analyst.

⇒
```
SELECT DNAME
FROM DEPT
WHERE DEPTNO IN (SELECT DEPTNO
                 FROM EMP
                 WHERE JOB = 'ANALYST');
```

Q.26. Write a query to display the employee name who is having maximum salary in Accounting department.

⇒
```
SELECT ENAME
FROM EMP
WHERE DEPTNO = ( SELECT DEPTNO
                 FROM DEPT
                 WHERE DNAME = 'ACCOUNTING' AND
                 SAL = ( SELECT MAX(SAL)
                         FROM EMP)) ;
```

Q.27. Write a query to list employee from RESEARCH & ACCOUNTING department having atleast 2 reporting

⇒
```
SELECT *
FROM EMP
WHERE DEPTNO IN (SELECT DEPTNO
                 FROM DEPT
                 WHERE DNAME IN ('RESEARCH',
                                 'ACCOUNTING');
```

Q.28. Write a query to display all the employee who located in chicago & there commission is zero.

⇒
```
SELECT ENAME
FROM EMP
WHERE DEPTNO = (SELECT DEPTNO
                FROM DEPT
                WHERE LOC = 'CHICAGO' AND
                      COMM = 0);
```

Q.29. Write a query to display department name of employee whose maximum salary & have no reporting manager.

⇒
```
SELECT DNAME
FROM DEPT
WHERE DEPTNO = (SELECT DEPTNO
                FROM EMP
                WHERE SAL = (SELECT MAX(SAL)
                             FROM EMP
                             WHERE MGR IS NULL;
```

Joins ⇒

→ Joins are used to retriere the data from multiple tables.

Types ⇒

(1). Cartesian join
(2). Inner join.
(3). Outer join. → left outer join, Right outer join, full outer join
(4). Self join.

(1). Cartesian join ⇒

→ Each & Every record in one table directly matches with each & every record of another table.

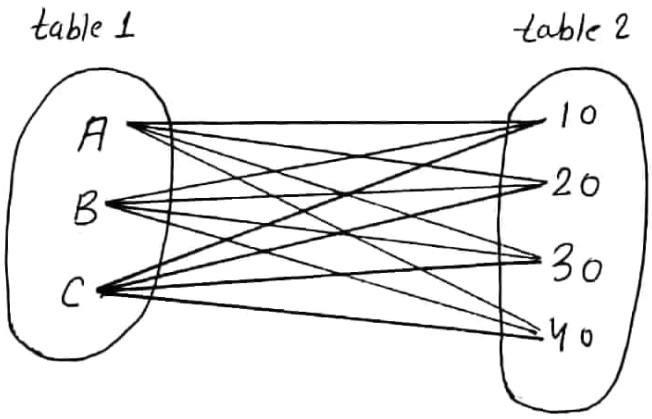
→ Here we are getting all valid & invalid outputs.

eg.

$A = \{A, B, C\}$

$B = \{10, 20, 30, 40\}$

$A*B$ or $(A,B) = \{(A,10), (A,20), (A,30), (A,40), (B,10), (B,20), (B,30), (B,40), (C,10), (C,20), (C,30), (C,40)\}$



eg.

```
SELECT ENAME, DNAME
FROM EMP, DEPT;
```

(2). Inner join ⇒

→ Inner join is used to display the matched record between the tables.

→ Each and Every record of one table is compared with each and Every record of another table and display the output of matched condition.

Oracle Syntax ⇒

```
SELECT t1.column_name , t2.column_name
FROM t1, t2
WHERE t1.common_column = t2.common_column;
```

ANSI syntax ⇒

```
SELECT t1.column_name, t2.Column_name
FROM t1 JOIN t2
ON t1.common_column = t2.common_column;
```

Q1. Write a query to retriere the departname of all the employee

⇒
```
SELECT ENAME, DNAME
FROM EMP, DEPT
WHERE EMP.DEPTNO = DEPT.DEPTNO;
```

OR

```
SELECT ENAME, DNAME
FROM EMP E, DEPT D
WHERE E.DEPTNO = D.DEPTNO;
```

Using ANSI syntax :

```
SELECT ENAME, DNAME
FROM EMP JOIN DEPT
ON EMP.DEPTNO = DEPT.DEPTNO;
```

OR

```
SELECT ENAME, DNAME
FROM EMP E JOIN DEPT D
ON E.DEPTNO = D.DEPTNO;
```

Q2 Write a query to display employee name, job, department name, location of all the manager & clerk who are working in the dept accounting & sales.

IG:- @purohit_17 ⇒
```
SELECT ENAME, JOB, DNAME, LOC
FROM EMP JOIN DEPT
ON EMP.DEPTNO = DEPT.DEPTNO AND JOB IN('MANAGER',
'CLERK') AND DNAME IN('ACCOUNTING', 'SALES');
```

Q.3. Write a query to display employee name, salary and department name of all the employees who are working as salesman and not located at DALLAS.

⇒
```
SELECT ENAME, SAL, DNAME
FROM EMP JOIN DEPT
ON EMP.DEPTNO = DEPT.DEPTNO AND
    JOB IN('SALESMAN') AND
    LOC != 'DALLAS';
```

Q.4. Write a query to select employee name and dept.name and location of all the employee who are working as clerk.

⇒
```
SELECT ENAME, DNAME, LOC
FROM EMP JOIN DEPT
ON EMP.DEPTNO = DEPT.DEPTNO AND JOB = 'CLERK';
```

Q.5. Write a query to display the region name, country name and city. Use table allias.

⇒
```
SELECT * FROM TAB;

SQL > CONN
Enter user-name : HR
Enter password : tiger
Connected
SQL > SELECT * FROM TAB;

SELECT REGION_NAME, COUNTRY_NAME, CITY
FROM REGIONS R, COUNTRIES C, LOCATIONS L
WHERE R.REGION_ID = C.REGION_ID AND
      C.COUNTRY_ID = L.COUNTRY_ID;
```

OR

```
SELECT REGION_NAME, COUNTRY_NAME, CITY
FROM REGIONS R JOIN COUNTRIES C
ON R.REGION_ID = C.REGION_ID JOIN
LOCATIONS L ON C.COUNTRY_ID = L.COUNTRY_ID;
```

(3). Outer Join ⇒

→ Outer join is used to display matched as well as unmatched records from the table.

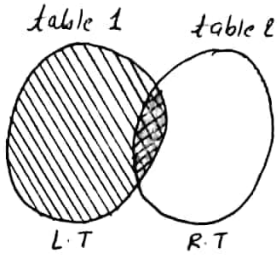
Outer join	=	Inner join	+	unmatched records

Types ⇒

(a). Left outer join.
(b). Right outer join.
(c). Full outer join.

(a). Left outer join ⇒

→ Left outer join is used to display the matched record between two tables along with the unmatched record from left table.



Oracle syntax:

```
SELECT t1.column_name, t2.column_name
FROM t1, t2
WHERE t1.common_column = t2.common_column(+);
```

ANSI syntax:

```
SELECT t1.column_name, t2.column_name
FROM t1 Left outer join t2
ON t1.common_column = t2.common_column;
```

Q. Write a query to display the employee name & department name of the employee.

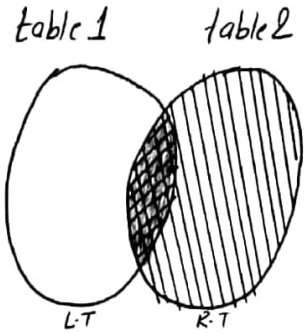
⇒
```
SELECT ENAME, DNAME
FROM EMP Left outer join DEPT
ON EMP.DEPTNO = DEPT.DEPTNO;
```

OR

```
SELECT ENAME, DNAME
FROM EMP, DEPT
WHERE EMP.DEPTNO = DEPTNO(+);
```

(b). Right outer join ⇒

→ Right outer join is used to display the matched record from both the tables along with the unmatched record from right table.



Oracle syntax ⇒

```
SELECT <column-list>
FROM t1, t2
WHERE t1.common_column (+) = t2.common_column;
```

ANSI syntax ⇒

```
SELECT <column-list>
FROM t1 Right outer join t2
ON t1.common_column = t2.common_column;
```

Q. Write a query to display the employee name & department name of the employees.

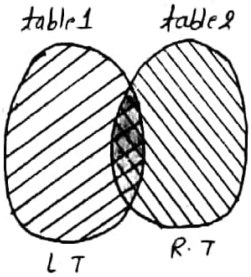
⇒
```
SELECT ENAME, DNAME
FROM EMP Right outer join DEPT
ON EMP.DEPTNO = DEPT.DEPTNO;
```

OR

```
SELECT ENAME, DNAME
FROM EMP, DEPT
WHERE EMP.DEPTNO (+) = DEPT.DEPTNO;
```

(c). Full - outer join ⇒

→ Full outer join is used to display both matched & unmatched record from both the table



Oracle syntax ⇒

```
SELECT <column_list>
FROM t1, t2
WHERE t1.common_column = t2.common_column (+)
UNION
SELECT <column_list>
FROM t1, t2
WHERE t1.common_column (+) = t2.common_column;
```

ANSI syntax ⇒

```
SELECT <column_list>
FROM t1 full outer join t2
ON t1.common_column = t2.common_column;
```

Q. Write a query to display the employee name & department name of the employee.

⇒

```
SELECT ENAME, DNAME
FROM EMP FULL OUTER JOIN DEPT
ON EMP.DEPTNO = DEPT.DEPTNO;
```

OR

```
SELECT ENAME, DNAME
FROM EMP, DEPT
WHERE EMP.DEPTNO = DEPT.DEPTNO (+)
UNION
SELECT ENAME, DNAME
FROM EMP, DEPT
WHERE EMP.DEPTNO (+) = DEPT.DEPTNO;
```

Q. Write a query to display employee name, salary and department name of all the employees who are working as salesman and not located at DALLAS.

⇒

```
SELECT ENAME, SAL, DNAME
FROM EMP FULL OUTER JOIN DEPT
ON EMP.DEPTNO = DEPT.DEPTNO AND
   JOB = 'SALESMAN' AND
   LOC != 'DALLAS';
```

(4). Self join ⇒

→ Self join is used to joining the table to itself

eg.
```
SELECT DEPTNO
FROM EMP, DEPT
WHERE EMP.DEPTNO = DEPT.DEPTNO;
```

error; 'Column ambiguously defined'.

```
SELECT EMP.DEPTNO
FROM EMP, DEPT
WHERE EMP.DEPTNO = DEPT.DEPTNO;
```

NOTE ⇒

→ If we want to display the common column of two tables, we need to give table name prior to the common column in SELECT statement.

eg.
```
SELECT DEPT.DEPTNO
FROM EMP, DEPT
WHERE EMP.DEPTNO = DEPT.DEPTNO;
```
→ Here, still it shows the duplicate values.

```
SELECT DISTINCT DEPT.DEPTNO
FROM EMP, DEPT
WHERE EMP.DEPTNO = DEPT.DEPTNO;
```

Q.1 Write a query to display all the employees who have manager.

⇒
```
SELECT A.ENAME EMPLOYEE, B.ENAME MANAGERS
FROM EMP A, EMP B
WHERE A.MGR = B.EMPNO;
```

Q.2. Write a query to display employees and managers name whose the employees join before their managers.

⇒
```
SELECT A.ENAME EMPY, B.ENAME MANAGER
FROM EMP A, EMP B
WHERE A.MGR = B.EMPNO AND A.HIREDATE < B.HIREDATE;
```

Q.3. Write a query to display all the emp who earn same salary.

⇒
```
SELECT E1.ENAME, E1.SAL
FROM EMP E1, EMP E2
WHERE E1.SAL = E2.SAL AND E2.ENAME <> E1.ENAME;
```

Co-related subqueries ⇒

→ Outer query works on the principal of subqueries & joins.

→ When the subqueries are dependent on the output of the outer query, we call that as co-related subqueries.

→ Here, outer query will be executed first, then the output of outer query will be the input for subqueries.

Q. Write a query to find second highest salary of employee.

⇒
```
SELECT A.SAL
FROM EMP A
WHERE 0 = (SELECT COUNT(B.SAL)          <- Highest Salary
           FROM EMP B
           WHERE A.SAL < B.SAL);
```

```
SELECT A.SAL
FROM EMP A
WHERE 1 = (SELECT COUNT(B.SAL)          <- Second highest Salary
           FROM EMP B
           WHERE A.SAL < B.SAL);
```

⋮

```
SELECT A.SAL
FROM EMP A
WHERE (N-1) = (SELECT COUNT(B.SAL)      <- Nth highest salary.
               FROM EMP
               WHERE A.SAL < B.SAL);
```

DDL ⇒

- → CREATE
- → ALTER
- → RENAME
- → TRUNCATE
- → DROP

(1). CREATE ⇒

→ To create a new table.

eg. CREATE TABLE STUD (SNO NUMBER(10) PRIMARY KEY, SNAME VARCHAR(25), BRANCH VARCHAR(25), SEM NUMBER(5), BOOKID NUMBER(10));

(2). ALTER ⇒

→ To add a column to the existing table

eg. ALTER TABLE STUD
ADD CNAME VARCHAR(20);

Syntax:

```
ALTER Table table_name
ADD new_column datatype;
```

[redacted]

→ To drop an existing table

eg. ALTER TABLE STUD
DROP COLUMN CNAME;

Syntax:

```
ALTER Table table_name
DROP COLUMN column_name;
```

[redacted]

→ To rename the column name.

Syntax:

```
ALTER Table table_name
RENAME COLUMN old_columnname to new_colname;
```

eg. ALTER TABLE EMP
RENAME COLUMN SAL TO SALARY;

(3). RENAME ⇒

→ This command is used to rename the table name.

Syntax:

```
RENAME old_tablename to New_table_name;
```

(4). TRUNCATE ⇒

→ If we use truncate command, all records will be deleted but structure of the table remains same.

Syntax:

```
TRUNCATE TABLE TABLE_NAME;
```

eg. TRUNCATE TABLE STUD;

(5). DROP ⇒

→ If we use DROP command, all records will be deleted along with the structure of the table.

Syntax:

```
DROP TABLE TABLE_NAME;
```

eg. DROP TABLE STUD;

NOTE:

*. Flashback ⇒

→ It is used used to restore the drop table.

Syntax:

```
FLASHBACK TABLE TABLE_NAME TO BEFORE DROP;
```

eg. Flashback table stud to before drop;

*. PURGE ⇒

→ PURGE command is used to delete table from recyclebin.

Syntax:

```
PURGE TABLE TABLE_NAME;
```

eg. PURGE TABLE STUD;

DML →

→ INSERT.
→ UPDATE.
→ DELETE

(1). INSERT →

→ It is used to insert values into the table.

Syntax:

```
INSERT INTO table_name (C1, C2, C3)
                values ('V1', 'V2', 'V3');
```

eg. INSERT INTO STUD (SNO, SNAME, SEM)
VALUES (01, 'YASH', 8);

```
INSERT INTO table_name values ('V1', 'V2', 'V3');
```

eg. INSERT INTO STUD VALUES (02, 'KUSH', 8);

(2). UPDATE →

Syntax

```
UPDATE table_name
 SET Cname = 'values_new'
 WHERE C_name = 'old_value';
```

eg. UPDATE EMP
SET SAL = 1500
WHERE SAL = 800;

(3). DELETE →

→ It is used to delete from the table.

eg. DELETE FROM STUD
WHERE SNAME = 'YASH';

```
DELETE FROM table_name
WHERE <Condition/s>;
```

→ Delete command delete the selective row.

TCL ⇒

→ COMMIT.
→ ROLLBACK.
→ SAVEPOINT.

(1). **COMMIT** ⇒
→ Used to save DML changes.

(2). **ROLLBACK** ⇒
→ To restore earlier values.
→ We can't use ROLLBACK to restore earlier values after using COMMIT command.

(3). **SAVEPOINT** ⇒
→ SAVEPOINT is a command used to save the statement names and marks the current point in the transaction.

```
ROLLBACK to SAVEPOINT SAVEPOINT_name;
```

```
INSERT INTO EMP(EMPNO,ENAME)
        VALUES (1, 'SCOTI');
SAVEPOINT S1;
INSERT INTO EMP(EMPNO, ENAME)
        VALUES (2, 'ABCD');
SAVEPOINT S2;
INSERT INTO EMP (EMPNO, ENAME)
        VALUES (3, 'MADHV');
SAVEPOINT S3;
ROLLBACK TO SAVEPOINT S2;
```

DCL ⇒

→ GRANT.
→ REVOKE.

(1). GRANT ⇒

→ this command is used to give permission access or priviledges to the user.

```
CONN
SHOW USER                    (Here USER is HR)
SELECT * FROM EMP;
```

Error: 'table or view does not exist'.

```
CONN
SHOW USER
GRANT ALL ON EMP TO HR
```

Syntax:

```
GRANT <Permission> ON tablename TO <usermame>;
```

```
CONN
SHOW USER
SELECT * FROM EMP;
```

Error: 'table or view does not exist'.

```
SELECT * FROM SCOTT.EMP;
```

(2). REVOKE ⇒

Syntax:

```
REVOKE <Permissim> ON Tablename FROM <username>;
```

```
CONN
SHOW USER
SELECT * FROM SCOTT.EMPLOYEE
```

Error: 'table or view does not exist'.

Normalization ⇒

→ Normalization is one of the database design technique.

→ Normalization is organised in such a way that it reduces redundancy and dependancy of data.

→ Here, larger tables are divided into smaller tables.
Those smaller tables are linked with relationship.

→ E.F Codd inrented normalization and the first name is 1NF (First normal form) and then he continued to extend theory to 2NF (second normal form) and then again extended to 3NF

→ Later E.F.Codd joined with Raymond.F.Boyce and both together proposed one more normal form called as BCNF i.e. Boyce-Codd-Normal form

Actor	Movies	Producer
Salman Khan	Maine Pyar kiya Tiger Zinda hai	Karan Zohar
Hrithik	Koi mil gya	Rakesh Roushon
Siddarth Malhotra	Student of the year EK villian	Karan johar
Prabhas	Bahuballi Bahuballi 2	Rajamaulli

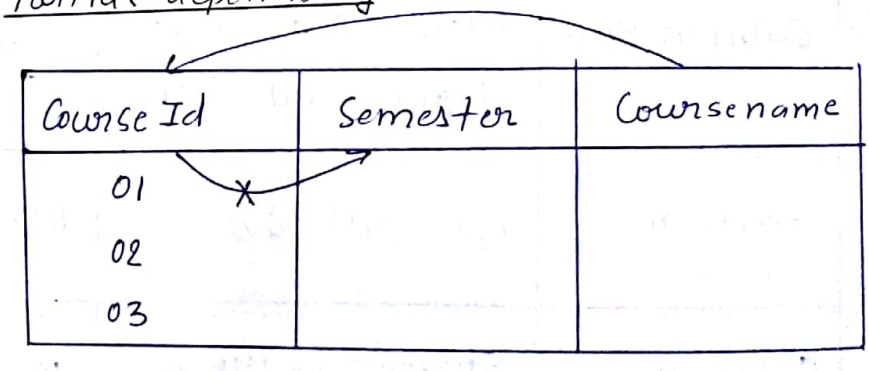
1NF ⇒

→ Each table cells should have single value.
→ Each records need to be unique.

Dependency ⇒ → Functional.
→ Partial.
→ Transitive.

Sr. no.	Actor	Movies	Producer
1.	Salman Khan	Maine Pyar kiya	Karan johar
2.	Salman Khan	Tiger Zinda hai	Karan johar
3.	Hrithik	Koi mil gya	Rakesh Roushan
4.	Siddarth Malhotra	Student of the year	Karan johar
5.	Siddarth Malhotra	EK villian	Karan johar
6.	Prabhas	Bahuballi	Rajamaulli
7	Prabhas	Bahuballi 2	Rajamaulli

(b). Partial dependency ⇒

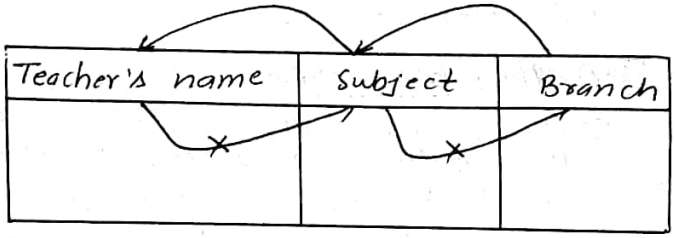


Here, in the above example, coursename is dependent on courseId but courseId doesnot dependent on semester.

2NF ⇒

→ Table should be in 1NF.
→ Table should have primary key.
→ There should be no partial dependancy.

(c). Transitive dependency ⇒



Here, in the above example Branch is dependent on subject and subject is dependent on Teacher's name but Teacher's name doesnot depend on subject and same for subject & branch.

3NF ⇒

→ Table should be in 2NF.
→ Table should not have transitive functional dependancy.

Anamolies ⇒

→ Not working according to requirements.

Types:
- → Insertion anamoly.
- → Updation anamoly.
- → Deletion anamoly.

(1). Insertion anamoly ⇒
- → let us consider an example suppose for a new admission until and unless a student opts a branch, data of student cannot be inserted or else we will have set the branch information as NULL.
- → Also if we have to insert data of 100 students of same branch then the branch information will be repeated for all those 100 students.
- → This is nothing but insertion anamolies.

(2). Updation anamoly ⇒
- → let me consider one example called as student table of the same branch, where the table consists of attributes like student name (sname), semester, HOD, department.
- → If the HOD leaves the college or is no longer the HOD of that department, then the records of all students have to be updated.
- → While updating, by mistake if we miss any record, it will lead to inconsistancy.
- → This is updatation anamoly.

(3). Deletion anamoly ⇒
- → let consider student table as an example, two different information are kept together, student information and branch information.
- → Hence at the end of academic year, if student records are deleted, we will loose the branch information.
- → This is deletion anamoly.

Views ⇒

Syntax:

```
CREATE VIEW VIEW_NAME
AS
(Subqueries);
```

eg. CREATE VIEW V1
AS
SELECT SNO, SNAME, PHNO FROM STUD1;

ROWNUM & ROWID ⇒

→ ROWNUM & ROWID are the pseudo-columns

ROWNUM ⇒

→ For each row RETURNBY a query, the ROWNUM pseudocolumn returns a number indicating the order in which oracle select the row from a table or set of joint rows

→ The first row selected has a row no. of 1

eg.

```
SELECT ROWNUM, EMP.*
FROM EMP;

SELECT ROWNUM, EMP.*
FROM EMP
WHERE JOB = 'CLERK';

SELECT *
FROM EMP
WHERE ROWNUM = 1;
```

= 2
>= 2
>= 3
} ✗

<= 2
<= 5
} ✓

Q. Write a query to display last four records.

⇒
```
SELECT * FROM EMP
MINUS
SELECT * FROM EMP
WHERE ROWNUM <= (SELECT COUNT(*)-4
                 FROM EMP);
```

ROWID ⇒

- → ROWID represents the physical address of the record.
- → ROWID consists of data-file as well as memory block address in the file.
- → ROWID will not being same for two or more records.

eg.
```
SELECT ROWID
FROM EMP;

SELECT ROWID, EMP.*
FROM EMP;

SELECT ROWID, EMP.*
FROM EMP
WHERE JOB = 'CLERK';
```