SQL NOTES

DATA:

Data is a raw fact that is used to describe the ATTRIBUTES of an ENTITY.

ATTRIBUTES: ATTRIBUTES are also known as properties.

ENTITY: Entity describes both living things and non-living things.

DATABASE:

A database is a place or a medium that is used to store the data in a systematic and organised manner.

In the database, we will be doing some basic operations.

a) **C –** Create / Insert.
b) **R –** Read / Retrieve.
c) **U** – Update / Modify.
d) **D –** Delete / Drop.

These are formally called ‘C R U D’ operations.

DBMS:-

DATABASE management system:

DBMS is software that is used to maintain and manage the database.

1) SECURITY AND AUTHORIZATION ARE THE TWO IMPORTANT FEATURES PROVIDED BY DBMS.
2) IN DBMS WE STORE THE DATA IN FILE FORMAT.
3) WE USE QUERY LANGUAGE TO COMMUNICATE WITH DBMS SOFTWARE.

RDBMS:

(RELATIONAL DATABASE MANAGEMENT SYSTEM)

RDBMS IS A SOFTWARE THAT IS USED TO MAINTAIN AND MANAGE THE DATABASE.

1) SECURITY AND AUTHORIZATION ARE THE TWO IMPORTANT FEATURES PROVIDED BY THE RDBMS.
2) RDBMS IS A TYPE OF DBMS SOFTWARE.
3) IN RDBMS WE STORE THE DATA IN TABLE FORMAT.
4) WE USE STRUCTURED QUERY LANGUAGE TO COMMUNICATE WITH RDBMS.
5) ANY DBMS SOFTWARE WHICH FOLLOWS RELATIONAL MODEL IS KNOWN AS ‘RDBMS’.

RELATIONAL MODEL:

1) RELATIONAL MODEL IS INVENTED BY A DATA SCIENTIST CALLED ‘E.F.CODD’.
2) HE GAVE THE MAIN RULE AS IN RDBMS THE DATA SHOULD BE STORED IN ‘TABLE FORMAT’.

TABLE:

THE LOGICAL ORGANIZATION OF ROWS AND COLUMNS IS CALLED A TABLE.

ROW:

ROW CAN ALSO BE CALLED AS RECORDS OR TUPLES.

COLUMN:

COLUMN IS ALSO KNOWN AS ATTRIBUTES OR FIELDS.

CELL:
1) THE SMALLEST UNIT OF A TABLE THAT IS USED TO STORE THE DATA IS KNOWN AS A CELL.
2) THE INTERSECTION OF ROWS AND COLUMNS CREATES A CELL.

RULES OF E F CODD:

1) THE DATA ENTERED INTO A CELL SHOULD BE 'SINGLE VALUE DATA'.
2) IN RDBMS THE DATA SHOULD BE STORED IN TABLE FORMAT, INCLUDING 'METADATA.
METADATA: THE DATA WHICH DESCRIBES ANOTHER DATA IS KNOWN AS METADATA, METADATA WILL BE STORED IN METATABLES.
3) ACCORDING TO E F CODD THE DATA CAN BE STORED IN DIFFERENT TABLES OR MULTIPLE TABLES, IF NEEDED WE CAN ESTABLISH A CONNECTION BETWEEN THE TWO TABLES, WITH THE HELP OF KEY ATTRIBUTES.
4) WE CAN VALIDATE THE DATA IN TWO STEPS
a) BY ASSIGNING DATATYPES.
b) BY ASSIGNING CONSTRAINTS.

NOTE: DATA TYPES ARE MANDATORY(COMPULSORY) BUT CONSTRAINTS ARE OPTIONAL.

DATA TYPES:

DATA TYPES ARE USED TO FIND OUT WHAT TYPE OR KIND OF DATA IS USED TO STORE IN A PARTICULAR COLUMN.

TYPES OF DATA TYPES:

1. CHAR:

CHAR DATA TYPE ACCEPTS ‘A-Z’, ‘a-z’, ‘0-9’ AND ‘@,#,$,!’.

Syntax:

CHAR(SIZE)

Whenever we use CHAR data type, we have to mention its size.

The maximum size can be 2000.

Char data type follows ***‘FIXED LENGTH MEMORY ALLOCATION’***

2. VARCHAR/VARCHAR2:

VARCHAR Data type accepts ‘A-Z’, ‘a-z’, ‘0-9’ and ‘!,@,#,$, etc’.

Syntax:

VARCHAR(SIZE)

Whenever we use VARCHAR data types, we have to mention their size.

In the VARCHAR data type, the maximum size can be 2000.

VARCHAR data type follows ‘***VARIABLE LENGTH MEMORY ALLOCATION’***.

2A) **VARCHAR2:** VARCHAR2 is an updated version of the VARCHAR data type.

In the VARCHAR2 data type, the maximum size can be 4000.

Syntax:

VARCHAR2(SIZE)

3. **DATE:** It is used to store dates in a particular column.

There are 2 formats:

Oracle-based formats:

‘DD-MON-YYYY’, and ‘DD-MON-YY’.

Syntax:

DATE

4. **NUMBER:** The Number data type is used to store numerical values in a particular column.

Syntax:
NUMBER(Precision,[scale])

Precision: Precision is used to determine the total number of digits present in a value.

Maximum precision can be 38.

Scale: A scale is used to determine the total number of decimal digits present in a value.

The maximum scale can be 127.

Scale is not mandatory and the default value of scale is ‘0’.

5. Large Objects:

i) **Character large objects**: Character large objects are used to store characters up to 4GB in size.

Syntax:

CLOB

ii) **Binary large objects**: Binary large objects are used to store binary numbers of images, documents, mp3, mp4, etc. up to 4GB in size.

Syntax:

BLOB

CONSTRAINTS:

Constraints are the additional validation that is given for a particular column.

Types of Constraints:

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

1) UNIQUE CONSTRAINT:

Unique is a constraint that is given for a column where the column should not accept duplicate or repeated values.

2) NOT NULL CONSTRAINT:

Not Null is a constraint that is given for a column where the column should not accept NULL Values.

b) NULL: NULL is nothing.

c) Zero is not a null value.

3) CHECK CONSTRAINT:

Check is a constraint that is an extra validation depending on conditions.

If the condition is true the value will be accepted, else if the condition is false it will be rejected.

Note: ***Whenever a column is being created we have to assign either NULL or NOT NULL constraint.***

4) PRIMARY KEY:

The primary key is a constraint that is used to identify a record uniquely from the table.

Characteristics of Primary key:

i) Primary key will not accept duplicate or repeated values.

ii) Primary key will not accept NULL values.

iii) Primary key is a combination of Unique and Not Null constraints.

iv) In a table we can have only one primary key.

v) Primary Key is not mandatory but recommended to have one in a table.

5) **FOREIGN KEY:**

A foreign key is a constraint that is used to establish a connection between 2 tables.

A foreign key is also known as ***'REFERENTIAL INTEGRITY CONSTRAINT'***

Characteristics of Foreign Key:

i) Foreign keys can accept duplicate or repeated values.

ii) Foreign keys can accept Null values.

iii) Foreign key is not a combination of Unique and Not Null constraints.

iv) In a table we can have any number of Foreign keys.

v) To be a Foreign key it should be a Primary key in its own table.

vi) Foreign key is present in the Child table but belongs to the Parent table.

OVERVIEW OF SQL STATEMENTS

1st language: **DATA DEFINITION LANGUAGE (*DDL*)**

2nd language: **DATA MANIPULATION LANGUAGE(*DML*)**

3rd language: **TRANSACTION CONTROLLED LANGUAGE(*TCL*)**

4th language: **DATA CONTROLLED LANGUAGE(DCL)**

5th language: **DATA QUERY LANGUAGE(*DQL*)**

DATA QUERY LANGUAGE:

The data query language is used to learn how to fetch the data from the database.

Statements in DQL:

1. SELECT
2. PROJECTION

3. SELECTION
4. JOINS

2) PROJECTION:

Projection is used to fetch the data from the table by selecting columns only.

Syntax:

SELECT */[DISTINCT] COL_NAME / EXPRESSION [ALIAS]

FROM TABLE_NAME;

ORDER OF EXECUTION:

1. FROM
2. SELECT

POINTS:

1. *FROM CLAUSE* STARTS THE EXECUTION.
2. FOR *FROM CLAUSE* WE PASS TABLE_NAME AS AN ARGUMENT.
3. *FROM CLAUSE* WILL GO TO THE DATABASE AND SEARCH FOR THE TABLE, IF THE TABLE IS PRESENT

THEN THE TABLE WILL BE KEPT UNDER EXECUTION, ELSE ***FROM CLAUSE*** WILL STOP THE EXECUTION AND THROW US AN ERROR MESSAGE.

4. *SELECT CLAUSE* WILL BE EXECUTED AFTER THE EXECUTION OF *FROM CLAUSE.*

5. FOR ***SELECT CLAUSE*** WE PASS 'ASTERISK' / ***,** COLUMN_NAME OR AN EXPRESSION AS AN ARGUMENT.

6. *SELECT CLAUSE* IS USED TO DISPLAY THE RESULT.

7. *SELECT CLAUSE* IS RESPONSIBLE TO PREPARE THE RESULT TABLE.

NOTE: *SQL IS NOT CASE SENSITIVE BUT LITERALS ARE CASE SENSITIVE.*

TAB IS A KEY WORD THAT IS USED TO DISPLAY THE NAMES OF THE TABLES PRESENT IN THE DATABASE.

i) ASTERISK (*): *IT IS USED TO SELECT ALL THE COLUMNS AND RECORDS PRESENT IN THE TABLE. IT IS USED TO DISPLAY THE ENTIRE TABLE.*

ii) SEMICOLON (;): *THIS SYMBOL IS USED TO DETERMINE THE END OF THE STATEMENT.*

*EX: SELECT **

FROM EMP;

*EX: SELECT **

FROM DEPT;

Expression:

Any statement which gives us the result is known as Expression.

Expression is always a combination of Operands and Operators.

In Operands we have 2 types:

I) COLUMN NAME

II) LITERALS (VALUES):

A. NUMBER

B. DATE

C. CHARACTER

Always DATE AND CHARACTER LITERALS should be written in uppercase letters and enclosed within single quotes.

In **Operators:**

+,*,-,/ etc.

ALIAS:

Alias is an alternate name given to a column_name or expression present in the result table.

We can assign ALIAS names with or without using the “AS” keyword.

ALIAS names must be a single word or string separated by a _ or enclosed within “ “.

Ex:-

Annual_Salary

“Annual Salary”.

Distinct Clause:

1. The distinct clause is used to remove the duplicate records present in the result table.
2. The distinct clause has to be used as the first Argument in the select clause.
3. Whenever we pass multiple arguments in the distinct clause it always removes a combination of columns that are duplicated.

SELECTION:

Selection is used to fetch the data from the table by selecting both columns and records together.

WHERE CLAUSE:

Syntax:

SELECT */[DISTINCT] COL_NAME/EXPRESSION [ALIAS]

FROM TABLE_NAME

WHERE <FILTER_CONDITIONS>;

ORDER OF EXECUTION:
1. FROM
2. WHERE - ROW BY ROW
3. SELECT - ROW BY ROW

i) Where clause is used to filter the records.

ii) Where clause executes Row by Row.

iii) Where clause executes after the execution of from clause.

iv) For the Where clause we pass filter_condition as an argument.

v) For the Where clause we can also pass multiple conditions using Logical Operators.

CONDITION: COL_NAME OPERATOR VALUE.

OPERATORS IN SQL:

1. Arithmetic Operators *(+,-,*,/)*
2. Relational Operators *(<, <=, >, >=)*
3. Comparison Operators *(=, !=)*
4. Concatenation Operator *(||)*
5. Logical Operators ***(AND, OR, NOT)***
6. Special Operators ***(IN, NOT IN, BETWEEN, NOT BETWEEN, IS, IS NOT, LIKE, NOT LIKE)***
7. Subquery Operators ***(ALL, ANY, EXIST, NOT EXIST)***

CONCATENATION OPERATOR:

Concatenation Operator is used to joining the given strings or columns.

LOGICAL OPERATORS:

i) AND OPERATOR:

AND Operator returns true if all the conditions are true or satisfied.

AND operator is used between the conditions.

ii) OR OPERATOR:

OR operator returns true even if any of the conditions are true.

OR operator is used between the conditions.

iii) NOT OPERATOR:

NOT operator is used as ‘Negation’ (opposite).

We can pass only one input.

If we pass true it will return False.

Similarly, if we pass false it will return True.

SPECIAL OPERATORS:

i) IN operator:

The IN operator is similar to the '=' operator.

IN operator is a multi-value operator which can accept multiple values on the right-hand side.

Syntax:

Col_name/Expression IN (V1, V2, V3,.....,Vn);

ii) NOT IN operator:

NOT IN operator is similar to IN operator where it will reject the value instead of selecting it.

Syntax:

Col_Name/Expression NOT IN (V1, V2, V3,....., Vn);

iii) BETWEEN OPERATOR:

Between operators is used whenever we have a range of values. Between operators works including the ranges.

In the Between operator, the ranges cannot be interchanged.

Syntax:

Col_Name/Expression BETWEEN lower range and higher range

iv) NOT BETWEEN OPERATOR:

Not Between Operator is similar to between operator where it will reject the value instead of selecting it.

Syntax:

Col_Name/Expression Not Between lower range and higher range

v) IS OPERATOR:

IS operator is used only to compare with NULL

Syntax:

Col_Name/Expression IS NULL

vi) IS NOT OPERATOR:

IS NOT operator is similar to IS operator where it will reject the values instead of selecting it.

Syntax: Col_Name/Expression IS NOT NULL

vii) LIKE OPERATOR:

Like Operator is used to perform pattern matching.

To achieve pattern matching we have to use the following special characters.

% - Any characters, Any number of characters, Any number of times characters are repeated, No character.

_(Underscore) - Accepts exactly only one character, it can be any character.

Note: % and _(underscore) are also called Wild Characters.

FUNCTIONS

FUNCTIONS are also called METHODS.

Functions are blocks of code or a set of instructions that are used to perform a particular task.

Functions/Methods:

a. User-defined.

b. Predefined / Built-In Functions:
i) Single Row Functions. SRF()
ii) Multi-Row Functions. MRF()

i) Single row functions:

a. Single row functions execute Row by Row.
b. It takes an input, starts the execution and generates the output, then it goes for the next input.
c. If we pass ‘n’ number of inputs to a Single row function it will return ‘n’ number of outputs.

ii) Multi-row functions:

a) Multi-Row functions execute group by group.

b. Multi-row functions are also known as **aggregate functions or group functions.**

c. If we pass ‘n’ number of inputs to a multi-row function it will return only one single output.

c. It takes all the input at once and combines it or aggregates it and gives us a single output.

List of Multi-Row functions:

a. MAX()

b. MIN()

c. SUM()

d. AVG()

e. COUNT()

NOTES:

a.Multi-Row functions can accept only a single argument which can be a Column_Name or Expression.

b.MAX() and MIN() can accept all the data type columns.

c. SUM() and AVG() can accept only a number datatype.

d.Multi-Row functions will ignore NULL values.

e. We cannot use Multi-row functions inside the where clause.

f. We cannot use any column name along with the multi-row functions in the select clause.

g. Count() is the only multi-row function that can accept '*'/ 'asterisk' as an argument.

Group By Clause

Syntax:

SELECT GROUP_FUNCTION / GROUP BY EXPRESSION

FROM TABLE_NAME

[**WHERE** <filter_condition>]

GROUP BY COL_NAME / EXPRESSION;

ORDER OF EXECUTION:

a. **FROM**

b. **WHERE** (if used) Row by Row

c. **GROUP BY** Row by Row

d. **SELECT** Group by Group

Points:

1. The group By clause is used to group the records.
2. The group by clause executes row by row.
3. Group By clause executes row by row but after the execution of group by clause, it will create groups.

4. Any clause which executes after the execution of group by clause will be executed group by group.
5. We can use group by expression along with the multi-row function in the select clause.

Group by Expression:

Any Col_name or an expression that is written inside Group by clause is known as a group by expression.

Having Clause:

Syntax:

SELECT GROUP_BY_EXPRESSION / GROUP FUNCTION

FROM TABLE NAME

[**WHERE** <FILTER_CONDITIONS>]

GROUP BY COL_NAME / EXPRESSION

HAVING <GROUP_FILTER_CONDITION>;

ORDER OF EXECUTION:

1. FROM
2. WHERE ROW BY ROW (IF USED)
3. GROUP BY ROW BY ROW
4. HAVING GROUP BY GROUP
5. SELECT GROUP BY GROUP

Points:

1. Having clause is used to filter the groups.
2. Having clause executes group by group.
3. Having clause executed after the execution of group by clause.

4. Since having executed after the group by clause, we have to write the group filter condition.
5. **Having clause cannot be used without using group by clause.**
6. We pass Group functions as arguments in the Having clause.

SUBQUERY

A query that is written inside another query is known as a subquery.

Points:
1. In the Subquery, we have a minimum of 2 queries.
a. Outerquery
b. Innerquery / Subquery.

2) In subquery the inner query starts the execution and gives you an output.

3) The output of Innerquery is given as an input to the outer query.

4) After getting the input from the inner query, the outer query starts the execution and generates the result.

5) Therefore we can say that the Outer query is dependent on Inner Query. (Because if outer query wants to execute the input is given by the Inner query)

Case 1: Whenever we have unknowns in the question we use a subquery.

Case 2: The data to be selected and the condition to be executed are present in two different tables then we use a subquery.

Types of Subquery.

1. Single Row Subquery
2. Multi-Row Subquery

1) Single Row subquery:

a. If a subquery returns only one record as an output then the subquery is known as a Single-row subquery.

b. For single-row subquery, we can use normal operators and special operators (IN, NOT IN)

2) Multi-Row subquery:

a. If a subquery returns more than one output then the subquery is known as a Multi-row subquery.

b. For the Multi-Row subquery, we should use Special operators (IN, NOT IN) and Subquery operators (ALL, ANY).

NOTE: For the Multi-Row subquery we cannot use normal operators.

Ex: Single-Row subquery:-

```
SELECT *
FROM EMP
WHERE DEPTNO = (SELECT DEPTNO
                FROM EMP
               WHERE DNAME = 'SALES');
```

Ex: Multi-Row subquery:-

```
 SELECT *
FROM EMP
WHERE DEPTNO IN (SELECT DEPTNO
                 FROM EMP
                 WHERE DNAME IN ('SALES',
'RESEARCH');
```

ALL OPERATOR:

ALL operator is a special operator which is used along with relational operator which will allow the values at the left-hand side to compare all the values present at the right-hand side.

Syntax:

Col_name/Expression < relational_operator > ALL (V1, V2, V3,....., Vn)

ANY OPERATOR:

ANY operator is a special operator which is used along with the relational operator which will allow the values present on the left-hand side to be compared with any one of the values present on the right-hand side.

Syntax:

Col_name/Expression < Relational_operator > ANY (V1, V2, V3,....., Vn)

Nested Subquery

A subquery that is written inside another subquery is known as a Nested subquery.

We can Nest up to 255 Subqueries.

Ex: WAQTD 3rd maximum salary.

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

EMPLOYEE-MANAGER RELATIONSHIP

a. Manager's name.

b. Work under.

c. Reporting to.

1. To find the employee's manager:

a. MGR of an employee.

b. MGR = EMPNO.

2) To find the employee's reporting to the given manager:

a. EMPNO of Manager.

b. EMPNO = MGR.

Ex: WAQTD BLAKE'S MANAGER NAME.

```
SELECT ENAME
FROM EMP
```

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

ORDER BY CLAUSE

SYNTAX:

```
SELECT COL_NAME
FROM TABLE_NAME
[WHERE <FILTER_CONDITION>]
[GROUP BY COL_NAME/EXPRESSION]
[HAVING <GROUP_FILTER_CONDITION>]
ORDER BY COL_NAME [ASC]/DESC;
```

ORDER OF EXECUTION:

```
FROM
[WHERE ROW BY ROW]
[GROUP BY ROW BY ROW]
[HAVING GROUP BY GROUP]
SELECT ROW BY ROW
ORDER BY [ASCENDING]/DESCENDING;
```

1. ORDER BY CLAUSE IS USED TO SORT THE RECORDS EITHER IN ASCENDING ORDER OR DESCENDING ORDER.
2. ORDER BY CLAUSE HAS TO BE WRITTEN AS THE LAST CLAUSE IN THE STATEMENT.
3. ORDER BY CLAUSE EXECUTES AFTER THE SELECT CLAUSE.
4. BY DEFAULT ORDER BY CLAUSE SORTS THE RECORD IN ASCENDING ORDER.

JOINS

The retrieval of data from multiple tables **simultaneously** is known as Joins.

Types of Joins:

a. CARTESIAN JOIN/CROSS JOIN.
b. INNER JOIN/EQUI JOIN.
c. OUTER JOIN:
1. LEFT OUTER JOIN
2. RIGHT OUTER JOIN
3. FULL OUTER JOIN

d) NATURAL JOIN

e) SELF JOIN

CARTESIAN JOIN:

A record from table one will be merged with all the records of table two.

The number of columns present in the result table will be the summation of no. of columns present in table 1 and table 2.

The number of records present in the result table will be the product of no. of records present in
table 1 and table 2.

Syntax:

ANSI:

```
SELECT COL_NAME
FROM TABLE 1 CROSS JOIN TABLE 2;
```

ORACLE:

```
SELECT COL_NAME
FROM TABLE 1, TABLE 2;
```

EX: FOR ANSI

```
SELECT *
FROM EMP CROSS JOIN DEPT;
```

EX: FOR ORACLE

```
SELECT *
FROM EMP, DEPT;
```

INNER JOIN / EQUI JOIN:

Inner join is used to obtain only the matched records.

Join condition:

It is a condition by which 2 tables are merged.

Syntax for Join Condition:

Table_name1.Col_name IN Table_name2.Col_name

Syntax ANSI:

Select Col_name

From Table_name1 INNER JOIN Table_name2

ON <Join Condition>;

Syntax Oracle:

Select Col_name

From Table_Name1, Table_Name2

Where <Join_Condition>;

Outer Join

Outer join is used to obtain only the unmatched records.

LEFT OUTER JOIN:

Left outer join is used to fetch the matched records along with unmatched records of the Left table.

Syntax:

ANSI:

Select Col_name

From Table_name1 LEFT [Outer] JOIN Table_name2

ON <Join Condition>;

Ex: ANSI

Select *

From EMP LEFT [Outer] JOIN DEPT

ON EMP.DEPTNO IN DEPT.DEPTNO;

Syntax:

Oracle:

```
Select Col_name
From Table_name1, Table_name2
Where Table_name1.Col_name IN Table_name2.Col_name(+);
```

Ex: Oracle

```
Select *
From EMP, DEPT
Where EMP.DEPTNO IN DEPT.DEPTNO(+);
```

Right Outer Join:

Right outer join is used to fetch the matched records along with unmatched records of the Right Table.

Syntax:

ANSI:

Select Col_name

From Table_name1 RIGHT [Outer] JOIN Table_name2

ON <Join_Condition>;

Ex: ANSI

SELECT *

FROM EMP RIGHT [OUTER] JOIN DEPT

ON EMP.DEPTNO IN DEPT.DEPTNO;

Syntax:

Oracle:

Select Col_name

From Table_name1, Table_name2

Where Table_name1.Col_name(+) IN Table_name2.Col_name;

Ex: ORACLE

SELECT *

FROM EMP, DEPT

WHERE EMP.DEPTNO(+) IN DEPT.DEPTNO;

FULL OUTER JOIN:

Full outer join is used to fetch the matched records along with the unmatched record of the left and right table.

Syntax:

ANSI:

SELECT COL_NAME

FROM TABLE_NAME1 FULL [OUTER] JOIN
TABLE_NAME2

ON <JOIN CONDITION>;

EX:

SELECT*

FROM EMP FULL [OUTER] JOIN DEPT

ON EMP.DEPTNO IN DEPT.DEPTNO;

NATURAL JOIN:

Syntax:

ANSI

SELECT COL_NAME

FROM TABLE_NAME1 NATURAL JOIN TABLE_NAME2;

a. While performing Natural join if there are common columns (Foreign key) present in the tables then it will give the output of INNER JOIN.

b. If there are no common columns present in the tables then it will give the output of Cartesian Join.

Note: In Natural join, we don't use any Join Condition.

Ex:

1. SELECT *

FROM EMP NATURAL JOIN DEPT;

2. SELECT *
FROM EMP NATURAL JOIN SALGRADE;

SELF JOIN:

Joining the same 2 tables or joining the table itself is known as Self Join.

Note:

Why do we use Self join?

Whenever we have different categories of data present in different records, if we want to display it in a Single record then we use Self Join.

Syntax:

ANSI:

```
SELECT COL_NAME
FROM TABLE_NAME T1 JOIN TABLE_NAME T2
ON <JOIN_CONDITION>;
```

Ex: ANSI

```
SELECT *
FROM EMP E1 JOIN EMP E2
ON E1.MGR IN E2.EMPNO
```

ORACLE:

```
SELECT COL_NAME
FROM TABLE_NAME T1, TABLE_NAME T2
WHERE T1.COL_NAME IN T2.COL_NAME;
```

NOTE:

ALIAS IS MANDATORY IN SELF JOIN FOR THE TABLE_NAMES.

ESCAPE CHARACTER

THE ESCAPE CHARACTER IS USED TO REMOVE THE SPECIAL BEHAVIOUR OF A SPECIAL CHARACTER AND IT TREATS IT AS A NORMAL CHARACTER.

THE ESCAPE CHARACTER IS USED TO CONVERT THE SPECIAL CHARACTER TO A NORMAL CHARACTER THAT IS NEXT TO IT, BUT ONLY ONCE.

THE ESCAPE CHARACTER IS WRITTEN BEFORE THE CHARACTER THAT IS TO BE TREATED AS AN ORDINARY CHARACTER.

CO - RELATED SUBQUERY:

a. In a Correlated subquery, the outer query starts the execution and generates partial o/p.

b. The partial o/p of the outer query will be given as an input to the inner query.

c. After getting the input from the outer query, the inner query starts the execution and generates the complete output.

d. The complete o/p of the inner query is given as an input to the outer query and the outer query starts the execution generating the result.

e. Therefore the outer query and inner query both are interdependent (depending on each other).

Note - 1) In the Correlated subquery join condition is mandatory, the join condition has to be written only in the Inner query.

2) Correlated subquery works with the principles of both Subquery and Joins.

EXISTS OPERATOR:

Exists is a subquery operator who can accept one operand towards the RHS.

Exists operator returns true if the subquery returns any value.

Exists operator is used to improving the efficiency of the Co-Related subquery.

NOT EXISTS OPERATOR:

Not exists returns true if the subquery returns NULL.

Difference between Subquery and Co-Related subquery.

SL No.	Subquery	Co-related Subquery

a)	The inner query starts the execution.	The outer query starts the execution.
b)	The outer query is dependent on Inner query.	Both Outer query and Inner query are interdependent.
c)	Join condition is not mandatory.	The join condition is mandatory and should be written in the Inner query.

1. To find Nth Maximum salary:

SYNTAX:

```
SELECT E1.SAL
FROM    EMP E1
WHERE (N-1) IN (SELECT COUNT(DISTINCT E2.SAL)
                FROM    EMP E2
                WHERE E1.SAL <  E2.SAL);
```

2. To find Nth Minimum salary:

SYNTAX:

```
SELECT E1.SAL
FROM     EMP E1
WHERE (N-1) IN (SELECT COUNT(DISTINCT E2.SAL
              FROM    EMP E2
              WHERE E1.SAL > E2.SAL);
```

SINGLE - ROW FUNCTIONS

1. MOD()
2. ROUND()
3. TRUNC()
4. LENGTH()
5. UPPER()
6. LOWER()
7. INITCAP()
8. REVERSE()
9. SUBSTR()
10. INSTR()

11. REPLACE()
12. TO-CHAR(), TO-DATE()
13. NVL()

NOTE: DUAL TABLE - Dual table is a dummy table that consists of a single record that is used to display our values in the result table.

1) MOD() Function:

This function is used to display the modulus of a given number.

SYNTAX:

MOD(M,N)

Ex: MOD(5,2)=1.

```
SELECT MOD(5,2)
FROM    DUAL;

MOD(5,2)
----------
```

1

2)ROUND() Function:

This function is used to round of the given number to its nearest value.

The round function can accept 1 argument.

SYNTAX:

ROUND(NUMBER)

Ex: 44.6 = 45.

Ex:

```
SELECT ROUND(55.5)
FROM DUAL;
```

```
ROUND(55.5)
-----------
         56
```

```
SELECT ROUND(88.2)
FROM DUAL;
```

```
ROUND(88.2)
-----------
         88
```

3) TRUNC() FUNCTION:

This function is similar to the round function but it always rounds the given number to its lowest value.

SYNTAX:
TRUNC(NUMBER)

Ex:

1. SELECT TRUNC(99.9)
   FROM DUAL;

TRUNC(99.9)

99

2. SELECT TRUNC(78.4)
   FROM DUAL:

TRUNC(78.4)
---

4) LENGTH() Function:

This function is used to obtain the Total number of Characters/digits given in the string/value.

SYNTAX:

LENGTH(‘STRING’)

Ex: LENGTH(‘QSPIDERS’) = 8

SELECT LENGTH(‘Q SPIDERS’)

FROM DUAL;

LENGTH(‘Q SPIDERS’)

9

SELECT LENGTH(ENAME), ENAME

FROM DUAL;

LENGTH(ENAME) ENAME

---

5 SMITH

5 ALLEN

4 WARD

5 JONES

6 MARTIN

---

5) UPPER() Function:

This function is used to convert the given STRING into UPPERCASE.

SYNTAX:

UPPER(‘STRING’)

Ex: UPPER(‘bangalore’) = BANGALORE

6) LOWER() Function:

This function is used to convert the given STRING into LOWERCASE.

SYNTAX:

LOWER(‘STRING’)

Ex: LOWER(‘BANGALORE’) = bangalore

7) INITCAP() Function:

This function is used to convert the given string Initial character to uppercase and other characters to lowercase.

SYNTAX:

INITCAP(‘STRING’)

Ex: INITCAP(‘bangalore’) = Bangalore

5) SELECT UPPER(‘bangalore’)

FROM DUAL;

UPPER(‘bangalore’)

BANGALORE

6) SELECT LOWER(‘BANGALORE’)

FROM DUAL;

LOWER(‘BANGALORE’)

---

bangalore

7) SELECT INITCAP(‘bangalore’)

FROM DUAL;

INITCAP(‘bangalore’)

Bangalore

8) REVERSE() Function:

This function is used to REVERSE the given string.

SYNTAX:

REVERSE(‘STRING’)

Ex: REVERSE(‘BANGALORE’) = EROLAGNAB

SELECT REVERSE(‘BANGALORE’)

FROM DUAL;

REVERSE(‘BANGALORE’)

EROLAGNAB

SELECT REVERSE(ENAME), ENAME

FROM EMP;

REVERSE(ENAME) ENAME

HTIMS SMITH

NELLA ALLEN

SMADA ADAMS

---

9) SUBSTR() Function:

This function is used to extract a part of a given string from the Original String.

SYNTAX:

SUBSTR('ORIGINAL STRING', POSITION, [LENGTH]);

10) INSTR() Function:

The instring function is used to obtain the position value of the substring if it is present in the original string.

SYNTAX:

INSTR(‘ORIGINAL STRING’, ‘SUBSTRING’, POSITION, [NTH OCCURRENCE])

NOTE: The default value of Nth occurrence is 1.

11) REPLACE() Function:

This function is used to replace a substring with a new string.

SYNTAX:

REPLACE(‘Original string’, ‘Substr’, ‘New String’)

12) TO_CHAR:

This function is used to convert the given date format into a string format.

SYNTAX:

TO_CHAR('DATE', 'FORMAT_MODELS')

FORMAT MODELS:

YEAR	TWENTY-TWENTY	DY	MON
YYYY	2020	DD	10
YY	20	D	2
MONTH	AUGUST	HH24	15

MON	AUG	HH12	3
MM	08	MI	5
DAY	MONDAY	SS	3

TO_DATE:

SYNTAX:

TO_DATE('DATE')

13) NVL(): [NULL VALUE LOGIC]

NVL() is used to overcome the drawbacks of operations on NULL.

SYNTAX:

NVL(ARG1, ARG2)

1. In ARG1 we write a col_name that can be a NULL, in ARG2 we write a value that will be substituted if the ARG1 is NULL.

2. If AGR1 is not NULL then the same value that is present in the ARG1 will be retained.

DATA DEFINITION LANGUAGE

THIS LANGUAGE IS USED TO CONSTRUCT A NEW OBJECT (TABLE/VIEW) OR MODIFY THE EXISTING OBJECT IN THE DATABASE.

STATEMENTS IN DDL:

1. CREATE
2. RENAME

3. ALTER
4. TRUNCATE
5. DROP

1. CREATE:

THIS STATEMENT IS USED TO CREATE A NEW TABLE IN THE DATABASE.

SYNTAX:

```
 CREATE TABLE   TABLE_NAME
(
COL_NAME 1 DATATYPE NOT NULL / [NULL],
COL_NAME 2 DATATYPE NOT NULL / [NULL],
…
…
,
COL_NAME N DATATYPE NOT NULL / [NULL],
```

CONSTRAINT CONS_REF_NAME UNIQUE(COL_NAME),

CONSTRAINT CONS_REF_NAME CHECK(CONDITION)

CONSTRAINT CONS_REF_NAME PRIMARY KEY(COL_NAME)

CONSTRAINT CONS_REF_NAME FOREIGN KEY(COL_NAME)

REFERENCES PARENT_TABLE_NAME(COL_NAME)

);

2) RENAME:

THIS STATEMENT IS USED TO CHANGE THE NAME OF THE EXISTING TABLE.

SYNTAX:
RENAME CURRENT_TABLE_NAME TO NEW_TABLE_NAME;

3) ALTER:

a. TO ADD A COLUMN:

```
ALTER TABLE TABLE_NAME
ADD COLUMN_NAME DATATYPE [NULL] / NOT NULL;
```

b. TO DROP A COLUMN:

```
ALTER TABLE TABLE_NAME
DROP COLUMN COLUMN_NAME;
```

c. TO CHANGE THE DATATYPE:

```
ALTER TABLE TABLE_NAME
MODIFY COLUMN_NAME NEW DATATYPE;
```

d. TO CHANGE THE NOT NULL / NULL CONSTRAINT:

```
ALTER TABLE TABLE_NAME
MODIFY COLUMN_NAME EXISTING DATATYPE
NULL / NOT NULL;
```

e. TO RENAME THE COLUMN:

```
ALTER TABLE TABLE_NAME
RENAME COLUMN CURENT_NAME TO NEW_NAME;
```

f. TO MODIFY CONSTRAINTS:

i) **ALTER TABLE TABLE_NAME**
ADD CONSTRAINT CONS_REF_NAME
UNIQUE(COL_NAME);

ii) **ALTER TABLE TABLE_NAME**
ADD CONSTRAINT CONS_REF_NAME
CHECK(CONDITION);

iii) **ALTER TABLE TABLE_NAME**
ADD CONSTRAINT CONS_REF_NAME
PRIMARY KEY (COL_NAME);

iv) **ALTER TABLE TABLE_NAME**
ADD CONSTRAINT CONS_REF_NAME
FOREIGN KEY (COL_NAME) REFERENCES
PARENT_TABLE_NAME (COL_NAME);

g. TO DROP/DISABLE/ENABLE A CONSTRAINT:

ALTER TABLE TABLE_NAME
DROP CONSTRAINT CONS_REF_NAME;
ENABLE CONSTRAINT CONS_REF_NAME;
DISABLE CONSTRAINT CONS_REF_NAME;

4) TRUNCATE:

TRUNCATE IS USED TO DELETE ALL THE RECORDS PERMANENTLY FROM THE TABLE, BUT THE TABLE

WILL NOT BE DELETED. (STRUCTURE OF THE TABLE REMAINS THE SAME)

SYNTAX:

TRUNCATE TABLE TABLE_NAME;

5) DROP:

SYNTAX:

DROP TABLE TABLE_NAME;

TO RECOVER THE TABLE (ONLY IN ORACLE)

SYNTAX:

FLASHBACK TABLE TABLE_NAME
TO BEFORE DROP
[RENAME TO NEW_NAME];

TO DROP THE TABLE FROM RECYCLE BIN

SYNTAX:

PURGE TABLE TABLE_NAME;

NOTE: DDL STATEMENTS ARE AUTO-COMMIT (AUTOSAVING)

DATA MANIPULATION LANGUAGE

THIS LANGUAGE IS USED TO INSERT, UPDATE AND DELETE RECORDS PRESENT IN THE TABLE.

STATEMENTS IN DML:

1) INSERT
2) UPDATE
3) DELETE

1) INSERT:

SYNTAX:

INSERT INTO TABLE_NAME VALUES (V1, V2,.., Vn);

EX:

a) INSERT INTO TEA VALUES (11, ‘ROSS’, 40000);
b) INSERT INTO TEA VALUES (12, ‘JOEY’, 35000);
c) INSERT INTO TEA VALUES (13, ‘CHANDLER’, 50000);

2) UPDATE:

SYNTAX:

UPDATE TABLE_NAME

SET COL1=V1, COL2=V2,….., COLn=Vn

[WHERE <FILTER_CONDITION>];

EX:

UPDATE TEA

SET SAL = 60000

WHERE TID = 13;

UPDATE TEA

SET SAL = 70000

WHERE TID = 12;

UPDATE TEA

SET SAL = 55000

WHERE TID = 11;

3) DELETE:

SYNTAX:

DELETE

FROM TABLE_NAME

[WHERE <FILTER_CONDITION>];

EX:

DELETE

FROM TEA

WHERE TID IN 11;

TRANSACTION CONTROLLED LANGUAGE

STATEMENTS IN TCL:

1) COMMIT:

SYNTAX:

COMMIT;

2) SAVEPOINT:

SYNTAX:

SAVEPOINT SAVEPOINT_NAME;

3) ROLLBACK:

SYNTAX:

ROLLBACK;

ROLLBACK TO SAVEPOINT:

SYNTAX:

ROLLBACK TO SAVEPOINT_NAME;

SAVEPOINT:-

SAVEPOINT IS USED TO CREATE THE CHECKPOINTS FOR THE RECORDS BEFORE SAVING THEM TO THE DATABASE.

ROLLBACK TO SAVEPOINT:

ROLLBACK TO SAVEPOINT IS USED TO ‘DO’ AND ‘UNDO’ PROCESSES FOR A PARTICULAR CHECKPOINT.

DATA CONTROL LANGUAGE

1) GRANT :

SYNTAX:

GRANT SQL_STATEMENT ON TABLE_NAME

TO USER_NAME;

GRANT STATEMENT IS USED TO GIVE PERMISSION TO ANOTHER USER ALONG WITH SOME RESTRICTIONS.

2) REVOKE :

SYNTAX:

REVOKE SQL_STATEMENT ON TABLE_NAME

FROM USER_NAME;

REVOKE IS USED TO TAKE BACK THE PERMISSION FROM THE USER ALONG WITH THE RESTRICTIONS. (TAKE BACK THE GIVEN PERMISSION)

ATTRIBUTES:-

1) KEY ATTRIBUTE / CANDIDATE KEY.
2) NON KEY ATTRIBUTE.
3) PRIME KEY ATTRIBUTE.
4) NON PRIME KEY ATTRIBUTE.

5) COMPOSITE KEY ATTRIBUTE.

6) SUPER KEY ATTRIBUTE.

7) FOREIGN KEY ATTRIBUTE.

1) KEY ATTRIBUTE:

AN ATTRIBUTE THAT IS USED TO IDENTIFY A RECORD UNIQUELY FROM THE TABLE IS KNOWN AS A KEY ATTRIBUTE.

2) NON KEY ATTRIBUTE:

ALL THE ATTRIBUTES EXCEPT THE KEY ATTRIBUTES IS KNOWN AS NON KEY ATTRIBUTE.

3) PRIME KEY ATTRIBUTE:

AMONG THE KEY ATTRIBUTES AN ATTRIBUTE CHOSEN TO BE THE MAIN ATTRIBUTE TO IDENTIFY THE RECORDS UNIQUELY FROM THE TABLE IS KNOWN AS THE PRIME KEY ATTRIBUTE.

4) NON PRIME KEY ATTRIBUTE:

ALL THE KEY ATTRIBUTES EXCEPT THE PRIME KEY ATTRIBUTE IS KNOWN AS NON-PRIME KEY ATTRIBUTE.

5) COMPOSITE KEY ATTRIBUTE:

IT IS A COMBINATION OF TWO OR MORE NON KEY ATTRIBUTES WHICH IS USED TO IDENTIFY A RECORD UNIQUELY FROM THE TABLE.

6) SUPER KEY ATTRIBUTE:

IT IS A SET OF ALL THE KEY ATTRIBUTES.

7) FOREIGN KEY ATTRIBUTE:

IT BEHAVES LIKE AN ATTRIBUTE OF ANOTHER ENTITY THAT IS USED TO REPRESENT THE RELATIONSHIP.

FUNCTIONAL DEPENDENCIES:

1) LET US CONSIDER A RELATION OF TWO ATTRIBUTES ‘X’ AND ‘Y’ RESPECTIVELY.

2) FOR EXAMPLE, LET US CONSIDER THAT ‘X’ DETERMINES ‘Y’, OR WE CAN ALSO SAY THAT ‘Y’ IS DEPENDENT ON ‘X’.

TYPES OF FUNCTIONAL DEPENDENCIES:

1) TOTAL FUNCTIONAL DEPENDENCY.

2) PARTIAL FUNCTIONAL DEPENDENCY.

3) TRANSITIVE FUNCTIONAL DEPENDENCY.

1) TOTAL FUNCTIONAL DEPENDENCY:

a) IF ALL THE ATTRIBUTES IN THE RELATION ARE DETERMINED BY A SINGLE ATTRIBUTE WHICH IS A KEY ATTRIBUTE AND THEN THERE exists TOTAL FUNCTIONAL DEPENDENCY.

b) IN TOTAL FUNCTIONAL DEPENDENCY WE DON’T HAVE ‘REDUNDANCY’ AND ‘ANOMALY’.

REDUNDANCY: THE UNWANTED REPEATATION OF DATA PRESENT IN THE TABLE IS KNOWN AS REDUNDANCY.

ANOMALY: THE SIDE EFFECTS THAT OCCURRED DURING ‘DML’ OPERATION (INSERT, UPDATE, DELETE) ARE KNOWN AS ANOMALIES.

Ex: LET US CONSIDER A RELATION OF 4 ATTRIBUTES,

A, B, C, D.

WHERE 'A' IS THE KEY ATTRIBUTE.

R-{A, B, C, D}

HERE 'A' IS A KEY ATTRIBUTE.

'A' DETERMINES 'B'

'A' DETERMINES 'C'

'A' DETERMINES 'D'

'A'-{'B', 'C', 'D'}

THEREFORE THERE EXISTS TOTAL FUNCTIONAL DEPENDENCY.

2) PARTIAL FUNCTIONAL DEPENDENCY:

a) FOR PARTIAL FUNCTIONAL DEPENDENCY TO EXIST THERE MUST BE A COMPOSITE KEY RELATION.

b) ONE OF THE KEY ATTRIBUTE IN THE COMPOSITE KEY RELATION DETERMINES ANOTHER ATTRIBUTE

SEPERATELY AND THEN THERE exist PARTIAL FUNCTIONAL DEPENDENCIES.

c) IN PARTIAL FUNCTIONAL DEPENDENCY WE HAVE REDUNDANCY AND ANOMALY.

EX: LET US CONSIDER A RELATION WITH 4 ATTRIBUTES ‘A, B, C, D’ IN WHICH ‘A & B’ ARE COMPOSITE KEY ATTRIBUTES.

R – {A, B, C, D}

A – D

BC

BD

THEREFORE THERE EXISTS PARTIAL FUNCTIONAL DEPENDENCY.

3) TRANSITIVE FUNCTIONAL DEPENDENCY:

a) AN ATTRIBUTE IS DETERMINED BY A NON KEY ATTRIBUTE WHICH IS INTERNALLY DETERMINED BY A KEY ATTRIBUTE THEN THERE EXISTS TRANSITIVE FUNCTIONAL DEPENDENCY.

b) IN TRANSITIVE FUNCTIONAL DEPENDENCY WE HAVE REDUNDANCY AND ANOMALY.

NORMALISATION

DECOMPOSITION OF LARGER TABLE INTO SEVERAL SMALLER TABLES TO REMOVE ‘REDUNDANCY’ AND ‘ANOMALY’ BY IDENTIFYING THEIR FUNCTIONAL DEPENDENCIES.

NORMAL FORM:

THE TABLE WITHOUT REDUNDANCY AND ANOMALY IS KNOWN AS THE NORMAL FORM.

LEVELS OF NORMAL FORM:

1) FIRST NORMAL FORM. (1^{ST} NF)

2) SECOND NORMAL FORM. (2^{ND} NF)

3) THIRD NORMAL FORM. (3^{RD} NF)

4) BOYCE – CODD NORMAL FORM. (BCNF)

NOTE:

IF THE TABLE IS REDUCED TO 3RD FORMAL FORM THEN WE CAN STATE THAT THE TABLE IS BEEN NORMALISED.

1) FIRST NORMAL FORM (1 NF):

A TABLE IS SAID TO BE IN THE FIRST NORMAL FORM, AND SHOULD SATISFY THE FOLLOWING CONDITION.

a) THE TABLE SHOULD NOT CONSIST MULTI-VALUE DATA.
b) THE TABLE SHOULD NOT CONTAIN DUPLICATE RECORDS.

2) SECOND NORMAL FORM (2 NF):

A TABLE IS SAID TO BE IN SECOND NORMAL FORM AND SHOULD SATISFY THE FOLLOWING CONDITION.

a) THE TABLE SHOULD BE NORMALISED TO 1ST NORMAL FORM.
b) THE TABLE SHOULD NOT HAVE PARTIAL FUNCTIONAL DEPENDENCY, IF IT CONTAINS THE RESPONSIBLE ATTRIBUTES WILL BE REMOVED FROM THE TABLE.

3) THIRD NORMAL FORM (3 NF):

A TABLE IS SAID TO BE IN THE THIRD NORMAL FORM AND SHOULD SATISFY THE FOLLOWING CONDITION.

a) THE TABLE SHOULD BE NORMALISED TO SECOND NORMAL FORM.
b) THE TABLE SHOULD NOT HAVE TRANSITIVE FUNCTIONAL DEPENDENCY, IF IT CONTAINS THE ATTRIBUTES WHICH ARE RESPONSIBLE WILL BE REMOVED FROM THE TABLE.

4) BOYCE - CODD NORMAL FORM:

a) IT IS AN UPDATED VERSION OF 3RD NORMAL FORM.
b) IT CAN ALSO BE CALLED AS 3.5 NF.
c) IT IS MORE STRICTER THAN 3RD NORMAL FORM.

PSEUDO COLUMNS:

These are the false columns that are present in every table. It should be called explicitly(should be called by the user).

We have 2 types of pseudo columns, they are:

1) ROWID.
2) ROWNUM.

ROWID:

Rowid will not be generated at first, because we will not have any rows.

It is generated when we insert some record.

Rowid is an 18-digit Hexa decimal address of the record which is stored in a particular memory location.

a) Rowid is Unique.
b) Rowid cannot be deleted or changed.
c) Rowid is generated at the time of insertion/creation of records.

   Therefore Rowid is ‘static’ in nature.

d) Rowid is present for each record.
e) Rowid is the fastest way to access or delete the record.

ROWNUM:

Rownum is used as a serial number which is assigned to the result table.

Rownum is dynamic, it keeps changing.

Rownum is generated at the time of result.

It is a serial number assigned to the result table.

a) Rownum always starts with (1).
b) Rownum gets incremented by (1) only.
c) Rownum is dynamic. (keeps changing)
d) Rownum is generated only when it is called.

How to make Rownum static?

1) Assign Rownum to a table and rename it as SLNO.
2) Use the above query in the form clause of the Outer query.
3) In the Where clause of the Outer query, write the suitable condition to get the result.

VIEW:

View is used to create a virtual table.

CONNECT SYSTEM / PASSWORD

GRANT CREATE ANY VIEW TO USER_NAME

Syntax:

CREATE VIEW VIEW_NAME

AS

SQL STATEMENT;