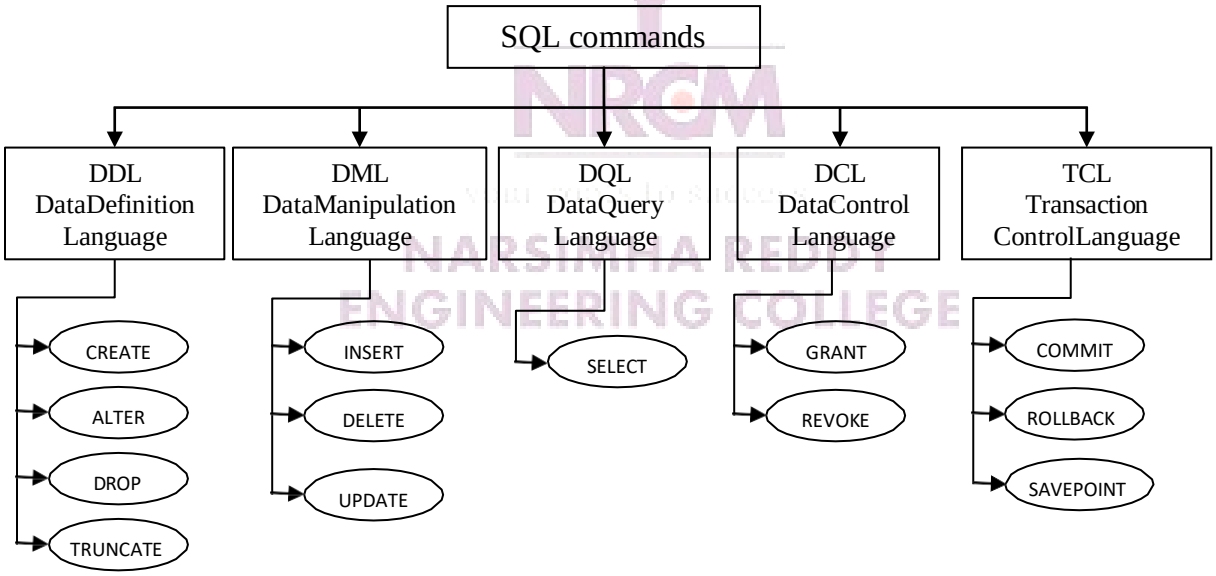
UNIT– II

SQL Queries and Constraints: Types of SQL Commands, Form of Basic SQL Query, SQL Operators, Aggregate Operators, Set Operators, Nested Queries, NULL values, Keys, Integrity Constraints Over Relations, Joins, Introduction to Views, Introduction to PL/SQL, Cursors, Triggers and Active Databases.

TYPES OF SQLCOMMANDS

Structured Query Language (SQL) is the database language used to create a database and to perform operations on the existing database. SQL commands are instructions used to communicate with the database to perform specific tasks and queries with data. These SQL commands are categorized into five categories as:

i. DDL: Data Definition Language
ii. DML: Data ManipulationLanguage
iii. DQL: Data QueryLanguage
iv. DCL: Data ControlLanguage
v. TCL: Transaction ControlLanguage.



i. **DDL(Data Definition Language) :**DDL or Data Definition Language consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structure ofdatabase objects in the database. The DQL commands are:

- **CREATE**: It is used to create the database or its objects (like table, index, function, views, store procedure and triggers).
- **DROP**: It is used to delete objects from the database.
- **ALTER**: It is used to alter the structure of the database.
- **TRUNCATE**: It is used to remove all records from a table, including all spaces allocated for the records are removed.

ii. **DQL (Data Query Language):** DML statements are used for performing queries on the data within schema objects. The purpose of DQL Command is to get data from some schema relation based on the query passed to it.The DQL commands are:

- **SELECT**–is used to retrieve data from the database.

iii. **DML (Data Manipulation Language):**The SQL commands that deals with the manipulation of data present in the database belong to DML or Data Manipulation Language and this includes most of the SQL statements. The DML commands are:

- **INSERT**–is used to insert data into a table.
- **UPDATE**– is used to update existing data with in a table.
- **DELETE**–is used to delete records from a database table.

iv. **DCL (Data Control Language):** DCL includes commands which mainly deal with the rights, permissions and other controls of the database system. The DCL commands are:

- **GRANT**-givesuser's access privileges to database.
- **REVOKE**-withdraw user's access privileges given by using the GRANT command.

v. **TCL (transaction Control Language):**TCL commands deals with the transaction with in the database. The TCL commands are:

- **COMMIT**–commitsa Transaction.
- **ROLLBACK**–rollbacks a transaction in case of any error occurs.
- **SAVEPOINT**–sets a save point within a transaction.

2. DDL COMMANDS

DDL or Data Definition Language consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structureof database objects in the database. The DQL commands are:

CREATE: It is used to create the database or its objects like table, index, function, views, store procedure and triggers.

a) **The 'CREATE DATABASE' Statement:** This statement is used to create a database.

```
Syntax: CREATE DATABASE Database_Name;
```

Example: CREATE DATABASEEmployee;

It creates Employee database.

b) **The 'CREATE TABLE' Statement:** This statement is used to create a table.

Syntax:

```
CREATE TABLE Table Name(
Column1 datatype(size)[column_constraint],
Column2 datatype(size)[column_constraint],
....
ColumnN datatype(size)[column_constraint],
[table_constraint]
[,table_constraint]
);
```

Note: *The content in the squarebrackets indicates it is optional. If not required ,you can skip it.*

Column constraints

- **PRIMARYKEY** //Use only, If one column name as primarykey.
- **NOTNULL** //It does not accept NULLvalue in that column.
- **DEFAULTvalue** //It storede fault value in that column, if no value is inserted
- **UNIQUE** //It allows to store only unique values in the column

Table constraints

- **PRIMARY KEY(column_name1,column_name2,…)**

Use it,If one column name or multiple column names acts as primarykey.

- **UNIQUE(column_name1,column_name2,…)**
  Use it,if one column name o rmultiple column names should contain unique values. If multiple column names are used, then for each row,it consider values from all the columns mentioned to decide the uniqueness, but not column wise.
- **FOREIGN KEY**(column_name1) **REFERENCES** other_table_name(column_name2) It is used to link data from one table to other table.
- **CHECK(condition)**
  It does not allow inserting value(s), if the condition is not satisfied. The condition may alsocontain multiple column names.

Example1: Creating table without any constraints

```
CREATE TABLE
Employee_Info (
    EmployeeID int, EmployeeName
    varchar(20), PhoneNumber
    numeric(10), City varchar(20),
    Country varchar(20)
);
```

Example2: Using PRIMARYKEY and NOTNULL as column constraints

```
CREATE TABLE
Departments (
    DeptID int PRIMARY KEY,
    DeptName varchar(20)NOTNULL,
    Hod varchar(20),
    Location varchar(20)
);
```

Example 3: Using PRIMARY KEY, NOT NULL, UNIQUE and DEFAULT as column constraints and FOREIGN KEY as table constraint.

```
CREATE TABLE
Students_Info (
    HallTicketNo int PRIMARYKEY,
    Name varchar(20)NOT NULL,
    Mobile numeric(10)NOTNULL UNIQUE,
    DepartmentID int,
    City varchar(20)DEFAULT‘Hyderabad’,
    FOREIGNKEY(DepartmentID)REFERENCES Departments(DeptID)
);
```

Example4: Using NOTNULL,UNIQUE as column constraints and PRIMARYKEYand CHECK as table constraints.

```
CREATE TABLE Voter_list
(
    VoterID numeric(10),
    Adhaar Nonumeric(12)NOTNULL
    UNIQUE, Name varchar(20)NOT NULL,
    Age int,
    Mobile numeric(10)UNIQUE,
    City varchar(20),
    PRIMARYKEY(VoterID),
    CHECK(AGE>18)
);
```

c) The 'CREATETABLEAS' Statement: You can also create a table from another existing table. The newly created table also contains data of existing table.

Syntax:

```
CREATE TABLE NewTableName AS(SELECT Column1,column2,...,ColumnN
FROM ExistingTableName
WHERE[condition]);
```

Example: CREATE TABLE Example Table AS(SELECT EmployeeName,PhoneNumber FROM Employee_Info);

ii. DROP: This statement is used to drop an existing table or a database.

a) The 'DROP DATABASE' Statement: This statement is used to drop an existing database. When you use this statement, complete information present in the database will be lost.

Syntax:
```
DROP DATABASE DatabaseName;
```

Example: DROP DATABASE Employee;

b) **The'DROPTABLE'Statement:** This statement is used to drop an existing table.When you use this statement, complete information present in the table will be lost.

Syntax:
```
DROP TABLE TableName;
```

Example: DROP TABLE Employee;

iii. TRUNCATE: This command is used to delete the information present in the table but does not delete the table. So, once you use this command, your information will be lost, but not the table.

Syntax:
```
TRUNCATE TABLE TableName;
```

Example: TRUNCATE TABLE Employee_Info;

iv. ALTER: This command is used to add, delete or modifycolumn(s) in an existing table. It can also be used to rename the existing table and also to rename the existing column name.

a) **The'ALTERTABLE' with ADD column:** You can use this command to add a new column to the existing table.

Syntax:
```
ALTER TABLE TableName ADD
Column Name Datatype;
```

Example: Adding Blood Group column to the Employee_Info table

```
ALTER TABLE Employee_Info
ADD Blood Group varchar(10);
```

b) **The 'ALTER TABLE' with DROP column:** You can use this command to remove acolumn from the existing table.

Syntax:
```
ALTER TABLE TableName
DROP ColumnName;
```

Example: Removing BloodGroup column from the Employee_Info table

```
ALTER TABLE Employee_Info
DROP BloodGroup;
```

c) **The 'ALTER TABLE' with MODIFY COLUMN:** This statement is used to change the data type or size of data type of an existing column in a table.

```
Syntax:    ALTER TABLE TableName
            MODIFY COLUMN ColumnName Datatype;
```

Example 1: Changing the size of column '*EmployeeName*' in table '*Employee_info*' from 20 to 30.

```
ALTER TABLE Employee_Info
MODIFY EmployeeName varchar(30);
```

Example 2: Changing the data type of column '*EmployeeID*' in the table '*Employee_info*' from *int* to *char(10)*.

```
ALTER TABLE Employee_Info
MODIFY Employee IDchar(10);
```

d) **The'ALTER TABLE' with CHANGE column name:** This statement is used to change the column name of an existing column in a table.

```
Syntax:    ALTER TABLE TableName
            CHANGE COLUMNOldColumnNameNewColumnName;
```

Example1: Changing the column name '*EmployeeName*' to '*EmpName*'in table '*Employee_info*'.

```
ALTER TABLE Employee_Info
CHANGE COLUMN EmployeeName EmpName;
```

e) **The 'ALTER TABLE' with RENAME table name:** This statement is used to change the table name in the database.

```
Syntax:    ALTER TABLE OldTableName
            RENAME TO NewTableName;
```

Example: Changing the table name from'*Employee_Info*'to '*Employee_Data*'.

```
ALTER TABLE Employee_Info
RENAME TO Employee_Data;
```

3. DML COMMANDS: The SQL commands that deals with the manipulation of data present in the database belong to DML or Data Manipulation Language and this includes most of the

SQL statements. The DML commands are:

i. **INSERT:** This statement is used to insert new record (row) into the table.

Syntax:
```
INSERT INTO TableName[(Column1,Column2,...,ColumnN)]
 VALUES (value1, value2,..., valueN);
```

Example1:

INSERT INTO Employee_Info(EmployeeID,EmployeeName,PhoneNumber,City,Country)
VALUES ('06', 'Sanjana', '9921321141', 'Chennai', 'India');

Example2: When inserting all column values as per their order in the table, you can omit column names.

INSERT INTO Employee_Info
VALUES('07','Sayantini','9934567654','Pune','India');

ii. **DELETE:** This statement is used to delete the existing records in a table.

Syntax:
```
DELETE FROM TableName
 WHERE Condition;
```

Example:

DELETE FROM Employee_Info
WHERE EmployeeName='Preeti';

Note: *If where condition is not used in DELETE command, then all the rows data will be deleted. If used only rows which satisfies the condition are deleted.*

iii. **UPDATE:** This statement isused to modify the record values already present in the table.

Syntax:
```
UPDATETableName
SETColumn1=Value1,Column2=Value2,... [WHERE
Condition];
```


Example:

UPDATE Employee_Info
SET EmployeeName='Jhon',City='Ahmedabad'
WHERE EmployeeID = 1;

Note:*If where condition is not used in UPDATE command, then in all the rows* Employee Name changes to 'Jhon' and City name changes to 'Ahmedabad'.*Ifusedonly rows which satisfies the condition are updated.*

4. DQL COMMAND: The purpose of DQL Command is to get data from one or more tables based on the query passed to it.

i. **SELECT:** This statement is used to select data from a database and the data returned isstored in a result table, called the **result-set**.

Syntax:

```
SELECT[DISTINCT]*/Column1,Column2,...ColumN
 FROM TableName
 [WHEREsearch_condition]
[GROUP BY column_names
 [HAVINGsearch_condition_for_GROUP_BY]
 [ORDER BY column_name ASC/DESC] ;
```

Example1:SELECT * FROM table_name;

Example2:

SELECT
EmployeeID,EmployeeName FROM
Employee_Info;

The 'SELECT with DISTINCT' Statement: This statement is used to display only different unique values. It mean it will not display duplicate values.

Example: SELECT DISTINCT PhoneNumber FROM Employee_Info;

The 'ORDER BY' Statement: The 'ORDER BY' statement is used to sort the required results in ascending or descending order. The results are sorted in ascending order by default. Yet, if you wish to get the required results in descending order, you have to use the **DESC** keyword.

Example

/*Selectallemployeesfromthe'Employee_Info'tablesortedby City*/

SELECT*FROMEmployee_Info
ORDER BY City;

/*Selectallemployeesfromthe'Employee_Info'tablesortedby City in Descending order */

SELECT*FROMEmployee_Info
ORDER BY City DESC;

/*Selectallemployeesfromthe'Employee_Info'tablesortedby City and EmployeeName. First it sort the rows as per city, then sort by employee name */

SELECT * FROM Employee_Info
ORDERBYCity,EmployeeName;

/*Selectallemployeesfromthe'Employee_Info'tablesortedby CityinDescendingorderandEmployeeNameinAscendingorder:*/

```
SELECT*FROMEmployee_Info
ORDERBYCityASC,EmployeeNameDESC;
```

Form of basic SQL query :

The basic form of an SQL query, specifically when retrieving data, is composed of a combination of clauses. Th most elementary form of an SQL query for data retrieval can be represented as

Syntax

```
SELECT [DISTINCT] column1, column2, ...
FROM tablename
WHERE condition;
```

The basic SQL query consists of three clauses:

1. SELECT Clause: This is where you specify the columns you want to retrieve. Use an asterisk (*) to retriev all columns.
2. FROM Clause: This specifies from which table or tables you want to retrieve the data.
3. WHERE Clause (optional): This allows you to filter the results based on a condition.
4. DISTINCT Clause (optional): is an optional keyword indicating that the answer should not contain duplicates. Normally if we write the SQL without DISTINCT operator then it does not eliminate the duplicates.

```
SELECT attribute_list
FROM relation_list
 WHERE condition;
```

Here are the primary components of SQL queries:

- SELECT: Retrieves data from one or more tables.
- FROM: Specifies the table from which you're retrieving the data.
- WHERE: Filters the results based on a condition.
- GROUP BY: Groups rows that have the same values in specified columns.
- HAVING: Filters the result of a GROUP BY.
- ORDER BY: Sorts the results in ascending or descending order.
- JOIN: Combines rows from two or more tables based on related columns.

AGGREGATE OPERATORS:

The SQL allows summarizing data through a set of functions called aggregate functions. The commonly used aggregate functions are: MIN(), MAX(), COUNT(), SUM(),AVG().

MIN() Function: The MIN function returns the smallest value of the selected column in a table.

Syntax:
```
SELECT MIN(ColumnName)
FROM TableName
WHERECondition;
```

Example:
```
SELECT MIN(EmployeeID) FROM
Employee_Info;
```

MAX( ) Function: The MAX function returns thelargest value of the selected column in a table.

Syntax:
```
SELECT MAX(ColumnName)
FROM TableName
WHERECondition;
```

Example:
```
SELECT MAX(Salary)ASLargestFees
FROM Employee_Salary;
```

COUNT()Function: The COUNT function returns the number of rows which match the specified criteria.

Syntax: SELECT COUNT(ColumnName) FROM TableName WHERECondition;

Example:
```
SELECT COUNT(EmployeeID)
FROM Employee_Info;
```

SUM() Function: The SUM function returns the total sum of a numeric column that you choose.

Syntax:
```
SELECT SUM(ColumnName)
FROM TableName
WHERECondition;
```

Example:
```
SELECT SUM(Salary)
FROMEmployee_Salary;
```

AVG() Function: The AVG function returns the average value of a numeric column that you choose.

Syntax:
```
SELECT
 AVG(ColumnName)
 FROM TableName
 WHERECondition;
```
Example:
```
SELECT AVG(Salary)
FROMEmployee_Salary;
```

The'GROUPBY'Statement: This 'GROUPBY 'statement is used with the aggregate functions to group the result-set by one or more columns.

Example:

```
--Tolistthenumberofemployeesfromeachcity.

SELECT COUNT(EmployeeID),City
FROM Employee_Info
GROUPBYCity;
```

The'HAVING'Clause: The'HAVING' clause must be used SQL along with GROUPBY clause only. It is similar to the **WHERE** clause.

Example

```
/*&amp;nbsp;&amp;nbsp;To list the number of employees in each city. The employees should be sorted high to low and only those cities must be included who have more than 5 employees:*/

SELECTCOUNT(EmployeeID),City
FROM Employee_Info
GROUPBY City
HAVINGCOUNT(EmployeeID)>2;
```

SQL Operators:

The different setof operators available in SQL are as follows:

- Arithmetic operators
- Bitwise operators
- Comparison operator
- Compound operator
- Logicaloperator

Letus look into each one of them, one by one.

Arithmetic Operators:

Operator	Description
%	Modulus [A %B]
/	Division [A /B]
*	Multiplication[A * B]
–	Subtraction[A– B]
+	Addition [A +B]

Bitwise Operators:

Operator	Description
^	BitwiseExclusiveOR(XOR) [A^ B]
|	BitwiseOR [A|B]
&	BitwiseAND[A &B]

Comparison Operators:

Operator	Description
<>	NotEqual to [A <>B]
<=	Lessthan or equalto [A<= B]
>=	Greaterthanor equalto[A>= B]
<	Lessthan [A<B]
>	Greaterthan[A>B]
=	Equalto [A =B]

Compound Operators:

Operator	Description
|*=	BitwiseORequals [A |*=B]
^-=	BitwiseExclusiveequals [A^-=B]
&=	BitwiseAND equals[A&= B]
%=	Moduloequals [A%=B]
/=	Divideequals [A /=B]
*=	Multiplyequals[A*= B]
-=	Subtractequals[A-= B]
+=	Addequals[A+= B]

Logical Operators: The Logical operators present in SQL are as follows:AND,OR,NOT, BETWEEN, LIKE, IN, EXISTS, ALL, ANY.

AND Operator: This operator is used to filter records that rely on more than one condition. This operator displays the records, which satisfy all the conditions separated by AND, and give the output TRUE.

Syntax:

```
SELECTColumn1,Column2,...,ColumnN
FROM TableName
WHERECondition1ANDCondition2ANDCondition3...;
```

Example:

```
SELECT*FROMEmployee_Info
WHERECity='Mumbai'ANDCity='Hyderabad';</pre>
```

OR Operator: This operator displays all those records which satisfy any of the conditions separated by OR and give the output TRUE.

Syntax:

```
SELECTColumn1,Column2,...,ColumnN
FROM TableName
WHERECondition1ORCondition2ORCondition3...;
```

Example:

```
SELECT*FROMEmployee_Info
WHERECity='Mumbai'ORCity='Hyderabad';
```

NOT Operator: The NOT operator is used, when you want to display the records which do not satisfy a condition.

Syntax:

```
SELECTColumn1,Column2,...,ColumnN
 FROM TableName
 WHERENOTCondition;
```

Example:

```
SELECT * FROM Employee_Info
WHERE NOT City='Mumbai';
```

NOTE: You can also combine the above three operators and write a query as follows:

```
SELECT * FROM Employee_Info
WHERE NOT Country='India' AND (City='Bangalore 'OR City='Hyderabad');
```

BETWEEN Operator: The BETWEEN operator is used, when you want to select values within a given range. Since this is an inclusive operator, both the starting and ending values are considered.

Syntax:

```
SELECTColumnN
ame(s) FROM
TableName
WHEREColumnNameBETWEENValue1ANDValue2;
```

Example:

```
SELECT * FROM Employee_Salary
WHERE Salary BETWEEN 40000 AND 50000;
```

LIKE Operator

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column of a table. There are mainly two wildcards that are used in conjunction with the LIKE operator:

- **%** :Itisused tomatches 0ormorecharacter.
- **_**:Itis usedtomatchesexactlyonecharacter.

Syntax

```
SELECTColumnName(s
) FROM TableName
WHEREColumnNameLIKEpattern;
```

Refer to the following table for the various patterns that you can mention with the LIKE operator.

LikeOperatorCondition	Description
WHERE CustomerName LIKE ‘v%	Finds any values that start with “v”
WHERE CustomerName LIKE ‘%v’	Finds any values that end with“v”
WHERE CustomerName LIKE ‘%and%’	Finds any values that have“and”in any position
WHERE CustomerName LIKE ‘_q%’	Finds any values that h0ave“q”in the second position.
WHERE CustomerName LIKE ‘u_%_%’	Finds any values that start with“u” and are at least 3 characters in length
WHERE ContactName LIKE ‘m%a’	Finds any values that start with“m”and end with “a”

Example:

```
SELECT * FROM Employee_Info
WHEREEmployeeNameLIKE'S%';
```

IN Operator: This operator is used for multiple OR conditions.This allows you to specify multiple values in a WHERE clause.

Syntax:SELECTColumnName(s) FROM
TableName

WHERE ColumnNameIN(Value1,Value2...);

Example:

```
SELECT * FROM Employee_Info
WHERE City IN('Mumbai','Bangalore','Hyderabad');
```

NOTE:You can also use IN while writing Nested Queries.

EXISTS Operator: The EXISTS operator is used to test if a record exists or not.

Syntax:

```
SELECTColumn
Name(s) FROM
TableName WHERE
EXISTS
(SELECT ColumnName FROM TableNameWHERE condition);
```

Example:

```
SELECT City
FROM Employee_Info
WHERE EXISTS (SELECTCity
                FROM Employee_Info
                WHERE EmployeeId=05 AND City='Kolkata');
```

ALL Operator: The ALL operator is used with a WHERE or HAVING clause and returns TRUEif all of the subquery values meet the condition.

Syntax:

```
SELECTColumnName(s)
    FROM TableName
    WHERE ColumnName operator ALL
        (SELECT ColumnName FROM TableName WHEREcondition);
```


Example:

```
SELECTEmployeeName
FROM Employee_Info
WHEREEmployeeID=ALL(SELECTEmployeeID
                    FROMEmployee_Info
                    WHERECity='Hyderabad');
```

ANYOperator: Similar to the ALL operator,the ANY operator is also used withaWHEREor HAVING clause and returns true if any of the subquery values meet the condition.

Syntax:

```
SELECTColumnN
ame(s) FROM
TableName
WHERE ColumnNameoperator ANY
(SELECTColumnNameFROMTableNameWHEREcondition);
```

Example:

```
SELECTEmployeeName
```

FROM Employee_Info

Aliases Statement: A liases are used to give a column /table a temporary name and only exists for duration of the query.

Syntax:/*AliasColumnSyntax.Insteadofdisplayingthecolumnname used in the table, it displays alias name. */

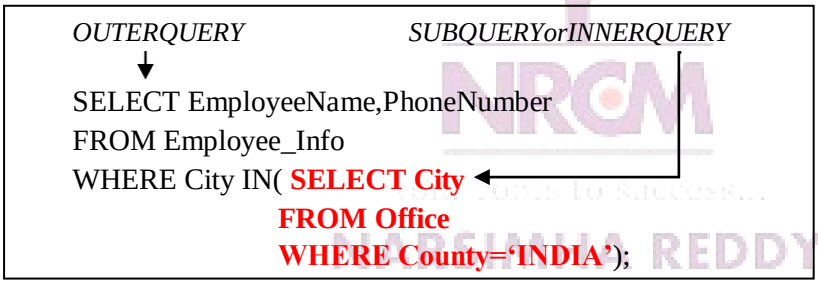
```
SELECTColumnNameASAliasName
FROM TableName;
```

Example:

```
SELECT Employee IDASID, Employee Name AS Emp Name
FROM Employee_Info;
```

NESTED QUERIES

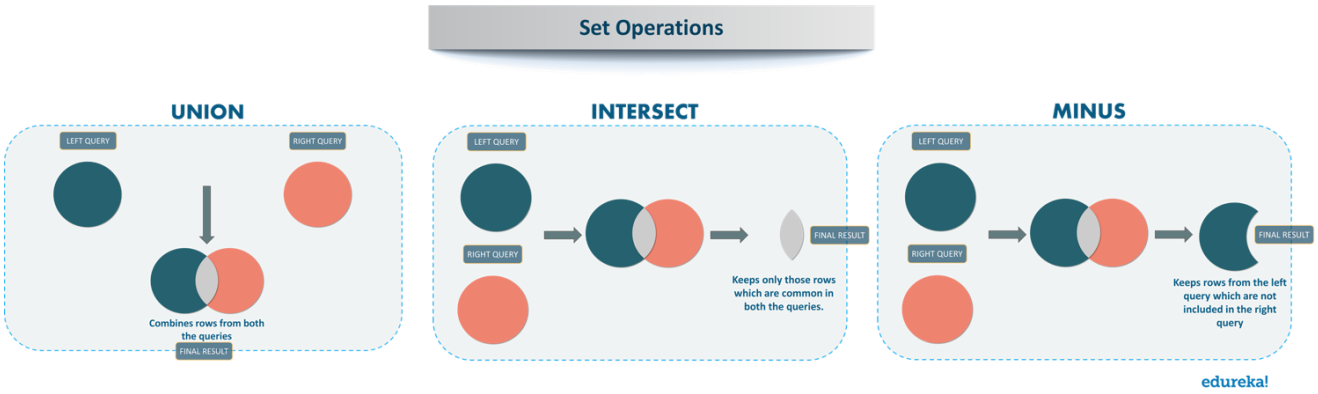
Nested queries are those queries which have an outer query and inner subquery. So, basically, the subquery is a query which is nested within another query.



First the inner query gets executed and the result will be used to execute the outer query.

SET OPERATORS: UNION, INTERSECT, EXCEPT

There are mainly three set operations: UNION, INTERSECT, EXCEPT.You can refer to the image below to understand the set operations in SQL.



ii. UNION: This operator is used to combine the result-set of two or more SELECT statements.

Syntax:
```
SELECTColumnName(s)FROMTable1WHERECondition UNION
SELECTColumnName(s)FROMTable2WHERECondition;
```

iii. INTERSECT: This clause used to combine two SELECTstatements and return the intersection of the data-sets of both the SELECT statements.

Syntax:
```
SELECTColumnName(s)FROMTable1WHERECondition
   INTERSECT
SELECTColumnName(s)FROMTable2WHERECondition;
```

iv. EXCEPT: This operator returns those tuples that are returned by the first SELECT operation, and are not returned by the second SELECT operation.

Syntax:
```
SELECT ColumnName(s)FROM Table1 WHERE condition
   EXCEPT
SELECT ColumnName(s)FROM Table2 WHERE condition;
```


Note: UNION, INTERSECT or EXCEPT operations are possible if and only if first SELECT query and second SELECT query produces same no of columns in same order, same column names and data type. Otherwise it gives an error.

INTEGRITY CONSTRAINTS OVER RELATIONS

Database Constraints are declarative integrity rules of defining table structures. They include the following 7 constraint types:

1. **Data type constraint:** This defines the type of data, data length, and a few other attributes which are specifically associated with the type of data in a column.
2. **Default constraint:** This defines what value the column should use when no value has been supplied explicitly when inserting a record in the table.
3. **Nullability constraint:** This defines that if a column is NOT NULL or allow NULL values

to be stored in it.

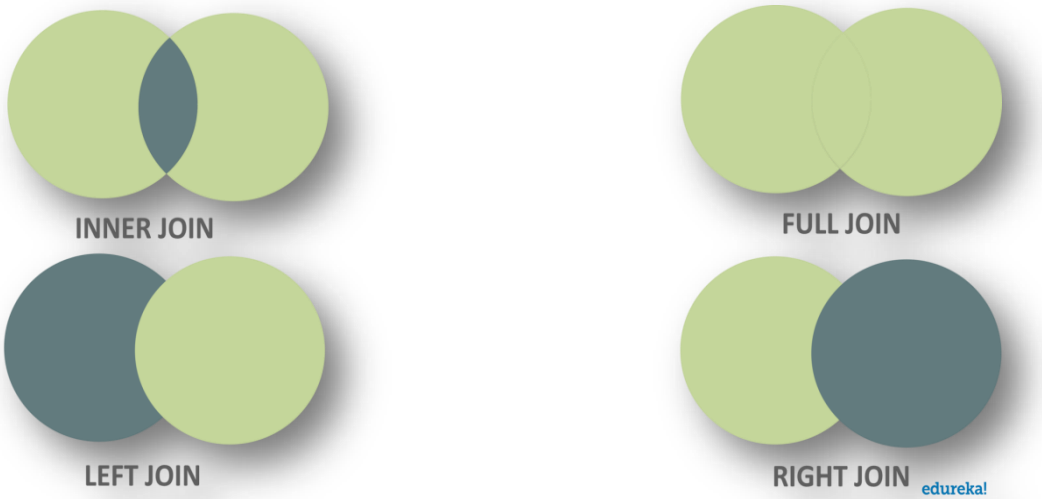
4. **Primary key constraint:** This is the unique identifier of the table. Each row must have a distinct value. The primary key can be either a sequentially incremented integer number or a natural selection of data that represents what is happening in the real world (e.g. Social Security Number). NULL values are not allowed in primary key values.
5. **Unique constraint:** This defines that the values in a column must be unique and no duplicates should be stored. Sometimes the data in a column must be unique even though the column does not act as Primary Key of the table. Only one of the values can be NULL.
6. **Foreign key constraint:** This defines how referential integrity is enforced between two tables.
7. **Check constraint:** This defines a validation rule for the data values in a column so it is a user-defined data integrity constraint. This rule is defined bythe user when designing the column in a table.

JOINS

JOINS are used to combine rows from two or more tables, based on a related column between those tables. The following are the types of joins:

- **INNER JOIN:** This join returns those records which have matching values in both thetables.
- **FULL JOIN:** This join returns all those records which either have a match in the left or the right table.
- **LEFT JOIN:** This join returns records from the left table, and also those records which satisfy the condition from the right table.
- **RIGHTJOIN:** Thisjoinreturnsrecordsfromtherighttable,andalsothoserecords which satisfy the condition from the left table.

Refer to the image below.



Let'sconsider the below Technologies and the Employee_Infotable, to understand the syntax of joins.

Employee_Info

EmployeeID	EmployeeName	PhoneNumber	City	Country
01	Shravya	9898765612	Mumbai	India
02	Vijay	9432156783	Delhi	India
03	Preeti	9764234519	Bangalore	India
04	Vijay	9966442211	Hyderabad	India
05	Manasa	9543176246	Kolkata	India

Technologies

TechID	EmpID	TechName	ProjectStartDate
1	01	DevOps	04-01-2019
2	03	Blockchain	06-07-2019
3	04	Python	01-03-2019
4	06	Java	10-10-2019

INNER JOIN or EQUI JOIN: This is a simple JOIN in which the result is based on matched data as per the equality condition specified in the SQL query. This join is used mostly. NATURAL JOIN is a type INNER JOIN. We can also use it. It also gives same result.

Syntax

```
SELECT
ColumnName(s) FROM
Table1
INNER JOIN Table2 ON Table1.ColumnName=Table2.ColumnName;
```

Example

```
SELECT T. TechID, E.Employee ID, E.EmployeeName
FROM Technologies T
INNER JOIN Employee_InfoEONT.EmpID=E.Emp ID;
```

TechID	EmployeeID	EmployeeName
1	01	Shravya
2	03	Preeti
3	04	Vijay

FULL OUTER JOIN: The full outer join returns a result-set table with the **matched data** of two table then remaining rows of both **left** table and r**ight** table with missing values are filled with NULL values.

Syntax

SELECTColumnName(s
) FROM Table1
FULLOUTERJOINTable2ONTable1.ColumnName=Table2.ColumnName;

Example

SELECTE.EmployeeID,E.EmployeeName,T.TechID
FROM Employee_Info E
FULLOUTERJOINTechnologiesTONE.EmployeeID=T.EmployeeID;

EmployeeID	EmployeeName	TechID
01	Shravya	1
02	Vijay	NULL
03	Preeti	2
04	Vijay	3
05	Manasa	NULL
06	NULL	4

LEFT JOIN: The left outer join returns a result-set table with the **matched data** from the two tables and then the remaining rows of the **left** table with null for the **right** table's columns.

Syntax:

SELECTColumnName(s)
FROM Table1
LEFTJOINTable2 ON Table1.ColumnName=Table2.ColumnName;

Example:

SELECTE.Employee Id, E.Employee Name,T.TechID
FROM Employee_Info E
LEFTJOINTechnologiesTONE.EmployeeID=T. Emp ID ID;

EmployeeID	EmployeeName	TechID
01	Shravya	1
02	Vijay	NULL
03	Preeti	2
04	Vijay	3
05	Manasa	NULL

RIGHT JOIN: The right outer join returns a result-set table with the matched data from the two tables being joined, then the remaining rows of the right table and null for the remaining left table's columns.

Syntax:

```
SELECTColumnName(s)
FROM Table1
RIGHTJOINTable2ONTable1.ColumnName=Table2.ColumnName;
```

Example:

```
SELECTE. Employee Id,E.EmployeeName,T.TechID
FROM Employee_Info E
RIGHTJOINTechnologiesTONE.EmployeeID=T.EmpID ID;
```

EmployeeID	EmployeeName	TechID
01	Shravya	1
03	Preeti	2
04	Vijay	3
NULL	NULL	4

INTRODUCTION TO VIEWS

A view is virtual tables whose rows are not explicitly stored in the database but are computed as needed from a view definition. They are used to restrict access to the database or to hide data complexity. A view contains rows and columns, just like a real table. Creating a view does not take any storage space as only the view query is stored in the data dictionary and the actual data is not stored.The tables referred in the views areknown as Base tables. Views do not contain data of their own. They take data from the base tables.

Thereasonsforusingviewsare

- Security is increased-sensitive information can be excluded from a view.
- Views can represent a subset of the data contained in a table.
- Views can join and simplify multiple tables into a single virtual table.
- Views take very little space to store; the database contains only the definition of a view, not a copy of all the data it presents.
- Different views can be created on the same basetable for different categories of users.

Creating Views syntax:

```
CREATE VIEW view_name AS
SELECT column_list
FROM table_name[WHERE condition];
```

Examples: Consider the below given employees table. Employees (eid, name,salary,experience)

employees

eid	ename	salary	Experience
101	Jhon	20000	2
105	Sam	18000	2
108	Ram	30000	4

If we want to hide the salary column from accessing a group of users, then we can create view on employees table as follows.

CREATE VIEW*emp* **AS**

SELECT*eid, ename, experience* **FROM** *employees*;

emp

eid	Name	Experience
101	Jhon	2
105	Sam	2
108	Ram	4

The view *emp* is a virtual table. The data in the *emp* table is not saved in the database but collectedfrom *employees* tablewhenever*emp*tableisreferredinSQLquery.Wecanperformall operations (INSERT, DELETE, UPDATE) on a view just like on a table but under some restrictions.

When can insertion, delete or update performed on view?

- The view is defined from one and only one table.
- The view must include the PRIMARY KEY of the base table.
- The base table columns which are not part of view should not have NOTNULL constraint.
- The view should not have any field made out of aggregate functions.
- The view must not have any DISTINCT clause in its definition.
- The view must not have any GROUPBY or HAVING clause in its definition.
- The view must not have any SUBQUERIES in its definitions.

i. **Inserting Rows into a View:** A new row can be inserted into a view in a similar way as you insert them in a table. When an insert operation performed on view, first a new row is inserted into the base table and the same is reflected in the view.

ii. **Deleting Rows into a View:** A row(s) can be deleted from a view in a similar way as you delete them from a table. When an delete operation performed on view, first row(s) is/are deleted from the base table and the same is reflected in the view.

iii. **Updating Rows into a View:** A row(s) can be updated in a view in a similar way as you update them in a table. When an update operation performed on view, first data is updated in the base table and the same is reflected in the view.

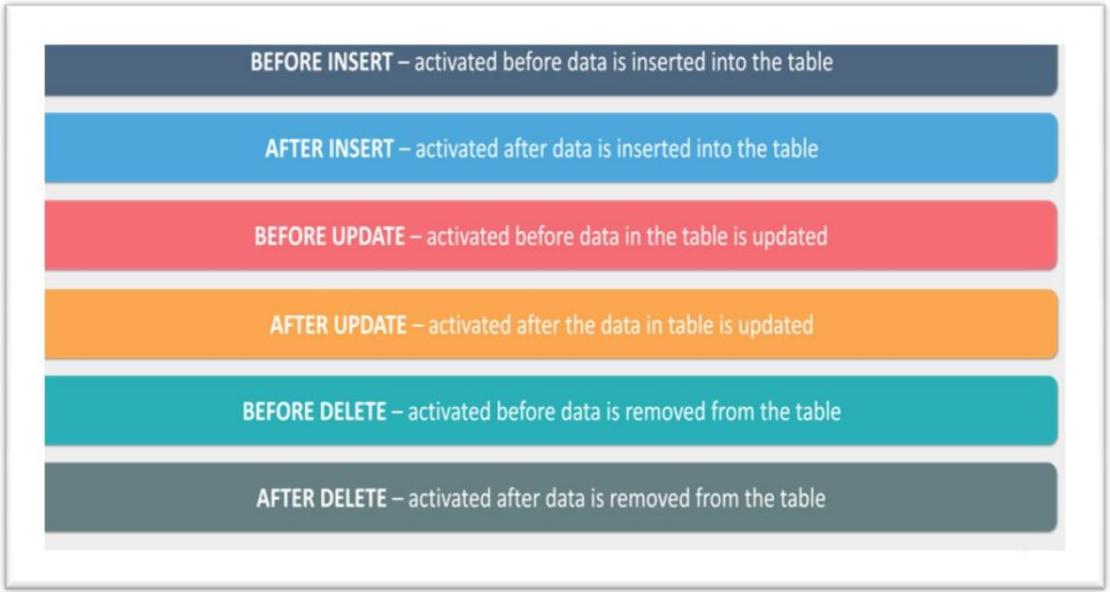
iv. **Dropping/Destroying View:** Whenever you do not need the view anymore, we can destroy the view by using DROP command. The syntax is very simple and is given below −

```
DROPVIEW view_name;
```

Example: DROP VIEW emp;

TRIGGERS

A trigger is a stored procedure in database which automatically invokes when ever a special event in the database occurs. For example, a trigger can be invoked when a row is inserted into a specified table or when certain table columns are being updated. So, a **trigger** can be invoked either **BEFORE** or **AFTER** the data is changed by**INSERT**, **UPDATE** or **DELETE**statement. Refer to the image below.



Syntax:

```
CREATE TRIGGER[TriggerName]
[BEFORE | AFTER]
{INSERT|UPDATE|DELETE}
on[TableName]
[FOREACHROW]
[TriggerBody]
```

Explanation of syntax:

- Create trigger[trigger_name]:Creates or replaces an existing trigger with the trigger_name.
- [before|after]:This specifies when the trigger will be executed.
- {insert|update|delete}: This specifies the DML operation.
- on[table_name]:This specifies the name of the table associated with the trigger.
- [foreachrow]:This specifies a row-level trigger,i.e.,the trigger will be executed for each row being affected.
- [trigger_body]:This provides the operation to be performed as trigger is fired.

BEFOREandAFTERofTrigger:

BEFORE triggers run the trigger action before the triggering statement is run.

AFTER triggers run the trigger action after the triggering statement is run.

EXAMPLE:

```
CREATE TRIGGER nb BEFORE INSERT ON accounts FOR EACH ROW /*Event*/
Begin
   IF:NEW.bal<0THEN                                          /*Condition*/
      DBMS_OUTPUT.PUT_LINE('BALANCE IS NAGATIVE..');         /*Action*/
   END IF;
End;
```

A trigger called '*nb*' is created to alert the user when inserting account details with negative balance value in to accounts table. Before inserting, the trigger is activated if the condition istrue. When a trigger activated, the action part of the trigger is get executed.

Active Databases

An active database is a database that uses triggers and other event-driven functionalities. The term "active" signifies that the DBMS reacts automatically to changes in data and predefined events. Triggers are a primary mechanism that makes a database "active."

Data Management

Key Features of Active Databases

1. **Event-Condition-Action (ECA) Rule:** This is the foundational concept of active databases. When a specific event occurs, the database checks a particular condition, and if that condition is met, an action is executed.
2. **Reactive Behavior:** The database can react to changes without external applications or users having to intervene, thanks to the ECA rules.
3. **Flexibility:** Active databases provide more flexibility in data management and ensure better data integrity and security.

Why are Active Databases Important?

- **Integrity Maintenance:** Active databases can enforce more complex business rules that can't be enforced using standard integrity constraints.
- **Automation:** They can automate certain tasks, reducing manual interventions.
- **Alerts:** They can notify users or applications when specific conditions are met.

Relation between Triggers and Active Databases

Triggers are what give an active database its "active" nature. The ability of the database to react to events automatically is primarily because of triggers that execute in response to these events.

Programming

In essence, while "trigger" refers to the specific procedural code blocks that run in response to events, "active database" refers to the broader capability of a DBMS to support and use such event-driven functionalities.

PROBLEMS

ConsiderarelationR isdecomposed intotwo sub relationsR_1andR_2.

- Ifall the followingconditions satisfy, thenthedecomposition is lossless.
- If anyof theseconditions fail, then thedecomposition is lossy.

Condition-01: Union of both the sub relations must contain all the attributes that are present in the original relation R.

$$R_1 \cup R_2 = R$$

Condition-02: Intersection of both the sub relations must not be null. In othe rwords, there must be some common attribute which is present in both the sub relations.

$$R1 \cap R2 \neq \emptyset$$

Condition-03: Intersection of both the sub relations must be a superkey of either R_1or R_2 or both.

****

Problem-01: Consider are lation schemaR(A,B,C,D)withthefunctionaldependencies A → B and C → D. Determine whether the decomposition of R into R_1 (A , B) and R_2 (C , D) is lossless or lossy.

Solution: To determine whether the decomposition is lossless or lossy, we will check all the conditions one by one. If any of the conditions fail, then the decomposition is lossy otherwise lossless.

Condition-01: According to condition-01,union ofboththe subrelations must containallthe attributes of relation R. So, we have:

$R_1(A, B) \cup R_2(C , D)$ = R(A , B, C , D)

Clearly, union of the sub relations contains all the attributes of relation R. Thus, condition-01 satisfies.

Condition-02: According to condition-02, intersection of both the sub relations must not be null. So, we have-

$R_1(A, B) \cap R_2(C , D)$ = Φ

Clearly, intersection of the sub relations is null. So, condition-02 fails. Thus, we conclude that the decomposition is lossy.

****

Problem-02: Consider a relation schemaR(A,B,C,D)withthefollowingfunctional dependencies

A→ B B→ C C → D D→ B

Determinewhetherthe decompositionof RintoR1(A , B), R2(B , C) and R3(B ,D)is lossless or lossy.

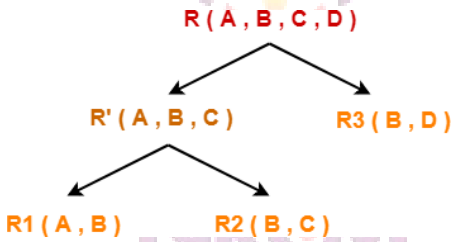
Solution:

StrategytoSolve: When a given relation is decomposed into more than two subrelations, then

- Considerany one possible ways in which the relation might have been decomposed into those sub relations.
- First, divide the givenrelationintotwosub relations.
- Then, divide the sub relations according to the sub relations given in thequestion.

As a thumb rule, remember-
Anyrelation can bedecomposed onlyinto two sub relations at a time.

ConsidertheoriginalrelationR wasdecomposedintothe givensubrelationsas shown:



Decomposition ofR(A,B, C,D) intoR'(A, B, C)and R_3(B,D)-

Todeterminewhetherthedecompositionis losslessor lossy,

- Wewill check all the conditions onebyone.
- Ifanyof theconditions fail, then thedecompositionis lossyotherwise lossless.

Condition-01: According to condition-01, union of both the sub relations must contain all the attributes of relation R. So, we have

R‘(A, B, C)∪R_3(B, D) = R (A , B, C , D)

Clearly, union of the sub relations contains all the attributes of relation R. Thus, condition-01 satisfies.

Condition-02 : According to condition-02, intersection of both the sub relations must not benull. So, we have

R‘(A, B, C)∩R_3(B,D) = B

Clearly, inter section of the sub relations is not null. Thus,condition-02 satisfies.

Condition-03: According to condition-03, intersection of both the sub relations must be thesuper key of one of the two sub relations or both. So, we have-

$R'(A, B, C) \cap R_3(B, D)$ = B

Now, the closure of attribute B is: $B^+=\{B,C,D\}$ So,

- Attribute'B'cannotdetermineattribute'A'ofsubrelation R'.
- Thus,itisnotasuper keyofthe sub relation R'.
- Attribute'B' candetermineall theattributesofsubrelation R_3.
- Thus,it is a super keyofthe sub relation R_3.

Clearly,inter section of the subrelations is a superkeyofoneofthesubrelations. So, condition-03 satisfies. Thus, we conclude that the decomposition is lossless.

Decomposition of R'(A, B,C) intoR_1(A, B)andR_2(B,C)-

Todeterminewhetherthedecompositionis losslessor lossy,

- Wewill check all the conditions onebyone.
- Ifanyof the conditions fail, then thedecomposition is lossyotherwise lossless.

Condition-01: According to condition-01, union of both the sub relations must contain all the attributes of relation R'. So, we have

$R_1(A, B) \cup R_2(B, C) = R'(A, B, C)$

Clearly, union of the sub relations contain all the attributes of relation R'. Thus, condition-01 satisfies.

Condition-02: According to condition-02, intersection of both the sub relations must not benull.

So, wehave

$R_1(A, B) \cap R_2(B, C) = B$

Clearly,inter section of the sub relations isnotnull.Thus,condition-02 satisfies.

Condition-03: According to condition-03, intersection of both the sub relations must be thesuper key of one of the two sub relations or both. So, we have

$R_1(A, B) \cap R_2(B, C) = B$

Now, the closure of attribute B is: $B^+=\{B,C,D\}$ So,

- Attribute'B'cannotdetermineattribute'A'ofsubrelation R_1.
- Thus,itis nota superkeyofthe sub relation R_1.
- Attribute'B' candetermineall theattributesofsubrelation R_2.
- Thus,it is a super keyofthe subrelation R_2.

Clearly, inter section of the sub relations is a super key of on eof the sub relations.So, condition-03 satisfies. Thus, we conclude that the decomposition is lossless.

5. CLOSURE OF AN ATTRIBUTE SET:

The set of all those attributes which can be functionally determined from an attribute set is called as a closure of that attribute set. Closure of attribute set {X} is denoted as $\{X\}^+$.

Steps to Find Closure of an Attribute Set: Following steps are followed to find the closure of an attribute set:

Step-01: Add the attributes contained in the attribute set for which closure is being calculated to the result set.

Step-02 : Recursively add the attributes to the result set which can be functionally determined from the attributes already contained in the result set.

Question1: Considerarelation R(A,B,C,D,E,F,G) withthefunctional dependencies

A→ BC, BC→DE, D→ F, CF→ G

Findtheclosureof{A}, {D} and{B,C}attributesandattributesets

Solution:

Closureof attributeA:

A^+= { A }

={ A , B, C } (UsingA→BC)

={ A , B, C , D , E} (UsingBC →DE)

={ A , B, C , D , E,F} (UsingD→F)

={ A , B, C , D , E,F,G } (UsingCF→ G)

Thus,

A^+={A , B , C , D, E ,F,G}

ClosureofattributeD:

D^+= { D }

={ D , F}(UsingD→ F)

Wecannotdetermineanyotherattributeusingattributes DandFcontainedintheresultset. Thus,

D^+= {D ,F}

Closureof attributeset{B, C}

{ B, C }$^+$={ B, C}

={ B, C , D , E} (UsingBC→DE)

={ B, C , D , E,F} (UsingD→F)

={ B, C , D , E,F, G } (UsingCF→ G)

Thus,

{B , C }$^+$= {B , C , D,E , F,G}

Question2: Consider the given functional dependencies

AB→ CD AF→ D DE→F C → G F→ E G→ A

Findtheclosureof {A,B}, {A, F},{B, G}and {C, F}

Solution:

Closureof {A, B}:

$\{AB\}^+$={ A ,B }

={ A , B, C , D } (UsingAB→ CD)

={ A , B, C , D ,G } (UsingC→ G)

Thus,$\{AB\}^+$={ A , B, C , D , G}

Closureof {C, F}:

$\{CF\}^+$={ C , F }

={ C , F, G } (UsingC→ G)

={ C , E , F, G } (Using F→ E)

={ A , C , E , E , F} (UsingG→A)

={ A , C , D , E , F,G } (UsingAF→ D)

Thus,{ $CF\}^+$={ A, C , D , E, F,G }

Closureof {B, G}:

$\{BG\}^+$={B, G }

={ A , B,G } (UsingG→A)

={ A , B, C , D ,G } (UsingAB→ CD)

Thus,{ $BG\}^+$={ A,B , C , D , G}

Closureof {B, G}:

$\{AF\}^+$={ A ,F }

={ A , D, F} (UsingAF→ D)

={ A , D, E ,F} (Using F→ E)

Thus,{ $AF\}^+$={ A ,D , E, F}

6. FINDING THE KEYS USING CLOSURE

Super Key:

Iftheclosureresultofanattributesetcontainsalltheattributesoftherelation, then that attribute set is called as a super key of that relation.

- Thus, we can say,**"Theclosureofasuperkeyistheentirerelationschema."**

Example: In the above example (Question 1),

- Theclosureof attribute A is theentirerelation schema.
- Thus , attribute A is asuper keyfor that relation.

CandidateKey:

- If there exists no subset of an attribute set whose closure contains all the attributes of the relation, then that attribute set is called as a candidate key of that relation.

Example: In the above example(Question1),

- NosubsetofattributeA contains alltheattributesofthe relation.
- Thus, attribute A is also a candidatekeyfor that relation.

Finding Candidate Keys From a Relation:

Wecan det erminethecandidatekeys ofa givenrelation usingthe followingsteps-

Step-01: Determine all essential attributes of the given relation

- EssentialattributesarethoseattributeswhicharenotpresentonRHSofanyfunctional dependency.
- EssentialattributesarealwaysapartofeverycandidatekeyThisisbecausethey cannot be determined by other attributes.

  Example: LetR(A,B,C,D,E,F)be are lation scheme with the following functional dependencies: A → B,C → DandD → E.

  The RHS of all the above functional dependencies contain only B, D and E. The attributes which are not present on RHS of any functional dependency are A, Cand F. So, essential attributes are: **A, C and F**.

Step-02: Determining all non-essential attributes using essential attributes

- TheattributesoftherelationthatarepresentinRHSarenon-essential attributes. They can be determined by using essential attributes.
- Now, following two cases are possible-
- **Case-01:** If all essential attributes together can determine all remaining on-essential attributes, then
  - Thecombinationofessentialattributesis thecandidatekey.
  - Itis the onlypossible candidatekey.
- **Case-02:** If all essential attributes together cannot determine all remaining on-essential attributes, then-

- Thesetofessentialattributesandsomenon-essentialattributeswillbethecandidate key(s).
- In this case, multiple can didatekeys are possible.
- To find thecandidate keys, we check different combinations of essential and non-essential attributes.

.

PROBLEMS ON FINDING CANDIDATE KEYS

Problem-01: LetR=(A,B,C,D,E,F)bearelationschemewiththefollowing dependencies:C→F,E→A,EC→DandA→B. Findthecandidatekey.Also, determine the total number of candidate keys and super keys.

Solution: We will find candidate keys of the given relation in the following steps-

Step-01: Determine all essential attributes of the given relation.

- Essential attributes of the relationare: C andE.
- So, attributes C and Ewill definitelybeapart ofeverycandidate key.

Step-02: Now, we will check if the essential attributes together can determine all remaining non-essential attributes. To check, we find the closure of CE. So,

$\{ CE \}^+$ ={ C , E }
={ C, E , F} (UsingC→F)
={ A , C , E , F} (UsingE→A)
={ A , C , D , E , F} (UsingEC→ D)
={ A , B, C , D , E,F} (UsingA→B)

We conclude that CEc an determine all the attributes of the given relation.So, **CE**istheonly possible candidate key of the relation.

TotalNumberofSuper Keys-

Therearetotal6attributes inthegiven relationofwhich-

- Thereare2essentialattributes-Cand E.
- Remaining4 attributesarenon-essential attributes.
- Essential attributes will be definitelypresent in every key.
- Non-essential attributesmayor maynot bepresent ineverysuperkey.

C E ABDF

Essential attributes Non-Essentialattributes

So, number of superkeys possible=2x 2x 2x 2=16.Thus, total number of superkeys possible = 16.

Problem-02: Consider the relation schemeR(E,F,G,H,I,J,K,L,M,N)and the set of functional dependencies: {E, F} → { G}, { F} → {I, J}, { E, H} → { K,L}, { K} → { M} and {L}→ {N}.Determine thecandidatekey(s).

Solution: We will find candidate keys of the given relation inthe following steps-

Step-01: Determine all essential attributes of the given relation.

- Essential attributesof the relationare-E, Fand H.
- So, attributes E, Fand Hwill definitelybeapart of every candidatekey.

Step-02:

- We will check if the essential attributes together can determine all remaining on-essential attributes.
- To check, we find the closure of

EFH. So, we have-

$\{EFH\}^{+}=\{E, F,H\}$
$=\{ E , F, G ,H \}$ (UsingEF → G)
$=\{ E , F, G ,H ,I, J\}$ (UsingF→IJ)
$=\{ E , F, G ,H ,I, J, K ,L\}$ (UsingEH→ KL)
$=\{ E , F, G ,H ,I, J, K ,L, M \}$ (Using K→ M)
$=\{ E , F, G,H,I, J,K,L, M, N\}$(UsingL→ N)

We conclude that EFH can determine all the attributes of the given relation. So, **EFH** is the only possible candidate key of the relation.

**********ALLTHE BEST **********