FUNDAMENTALS OF SQL

Overview of sql

INTRODUCTION TO SQL

DATA - Raw facts or unprocessed facts are called Data . Data has two Types , They are Qualitative and Quantitative.

INFORMATION - Processed Data is called Information.

SCHEMA OR DATABASE - Schema or Databases is an organized information, or data, typically stored in a table form.

SQL - Its a structured query language which is used to communicate with the database .

ER-DIAGRAM - Its an Entity Relationship diagram which represents visual representation of the table's structure and the Relationship between them.

COMPONENTS OF ER-DIAGRAM

ENTITY

It is an object that exists

- Weak Entity
- Strong Entity

ATTRIBUTES

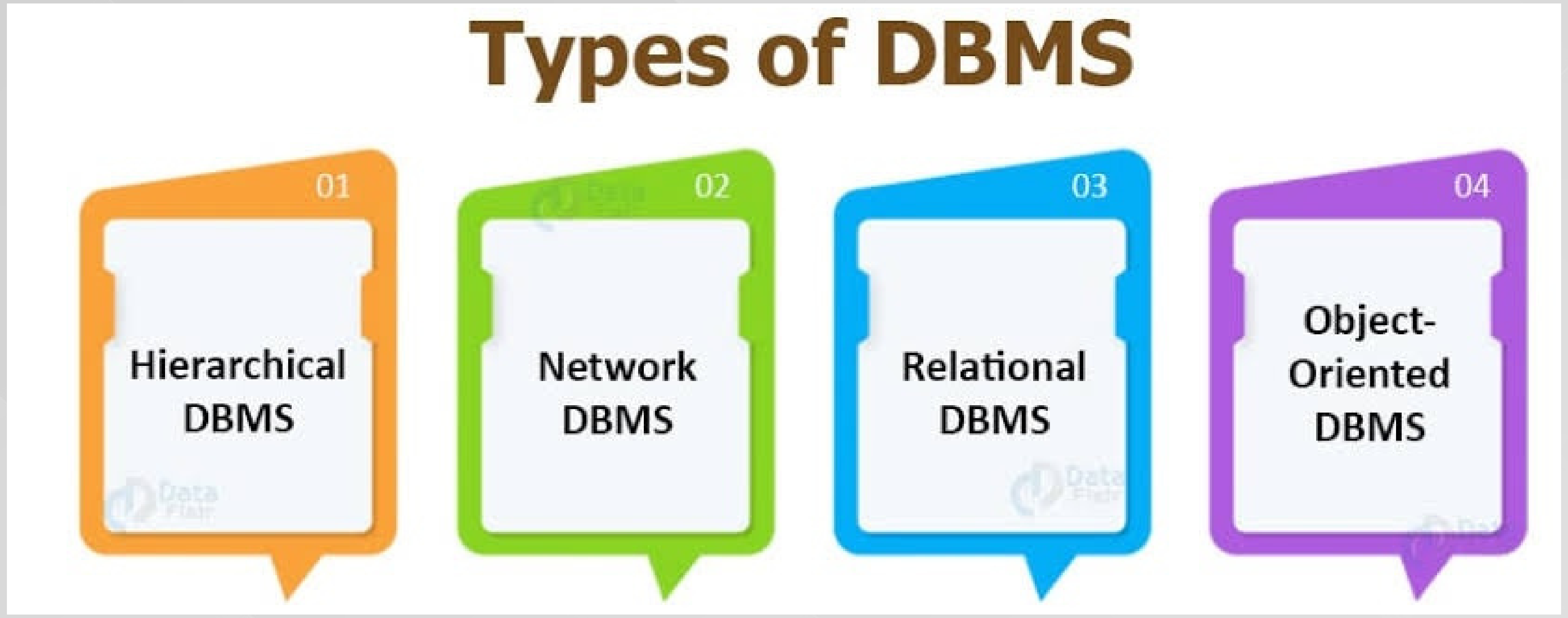
Properties of an object

- Key Attribute
- Multivalued Attribute
- Composite Attribute
- Derived Attribute

RELATIONSHIP

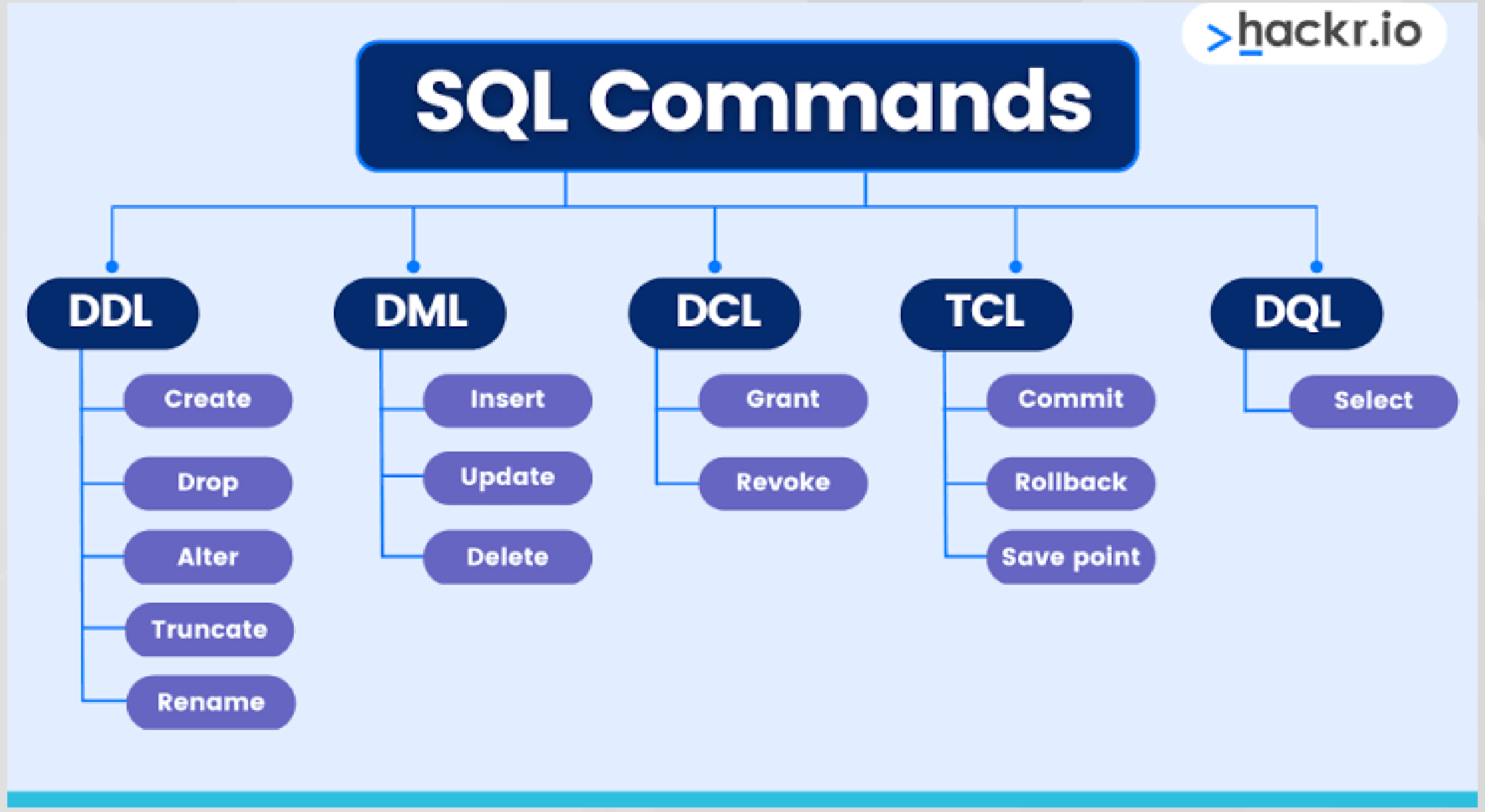
Relationship between entity and attributes

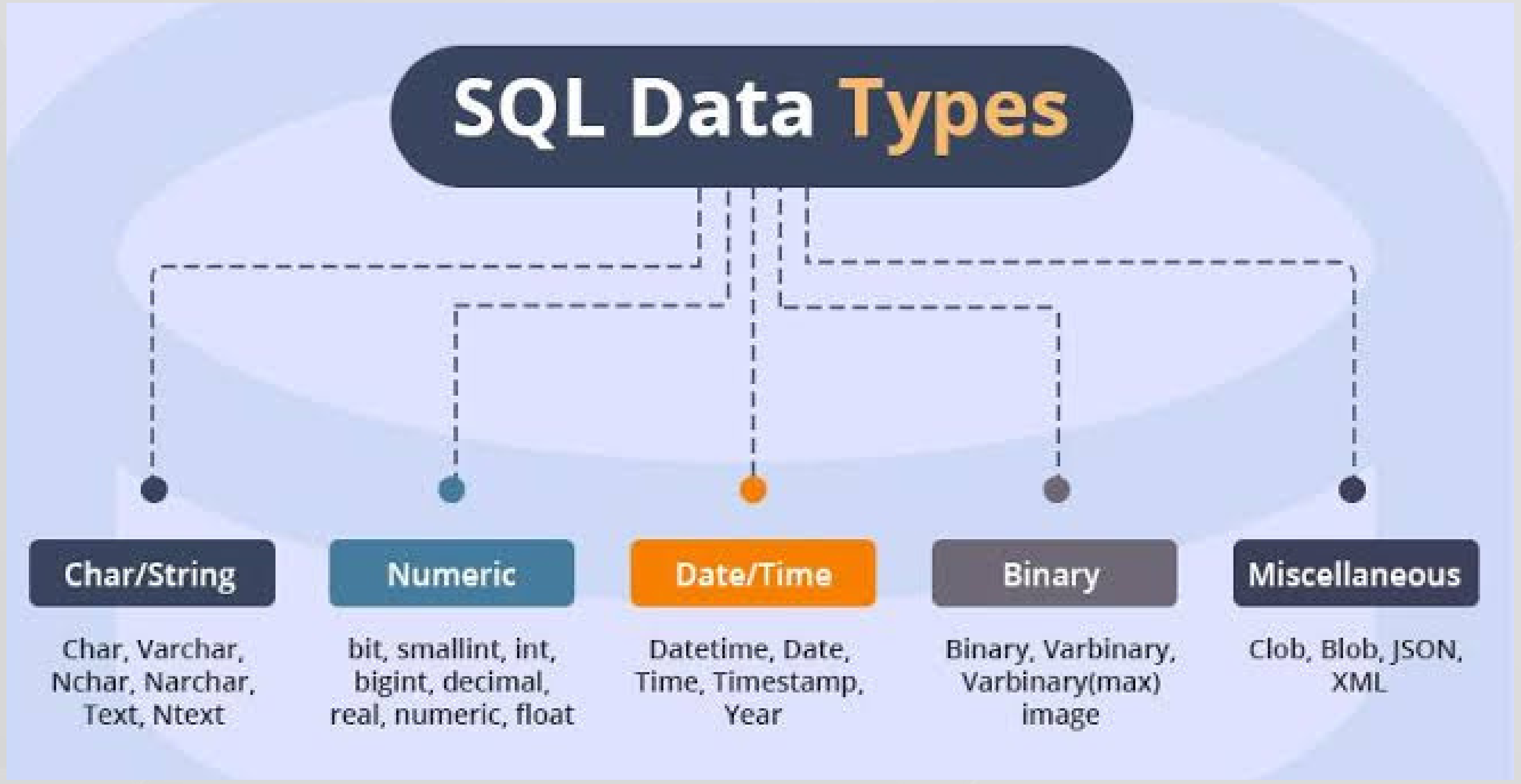
- One to One
- One to Many
- Many to One
- Many to Many



TYPES OF DBMS

TYPES OF SQL COMMANDS




SQL Data Types
Char/String
Char, Varchar, Nchar, Narchar, Text, Ntext
Numeric
bit, smallint, int, bigint, decimal, real, numeric, float
Date/Time
Datetime, Date, Time, Timestamp, Year
Binary
Binary, Varbinary, Varbinary(max) image
Miscellaneous
Clob, Blob, JSON, XML

SQL DATA TYPES

MANAGING DATABASE COMMANDS

CREATE DATABASE : Always the database name should be unique within the RDBMS

SYNTAX : CREATE DATABASE DatabaseName;

SELECT DATABASE , USE DATABASE :

The SQL USE statement is used to select any existing database in the SQL schema .

SYNTAX : USE DatabaseName;

DROP OR DELETE DATABASE :

The SQL DROP DATABASE statement is used to delete any existing database in the SQL schema .

SYNTAX : DROP DATABASE DatabaseName;

EXAMPLE :

```
CREATE DATABASE   Student;
USE    Student;
DROP DATABASE   Student;
```

DDL COMMANDS - DATA DEFINITION LANGUAGE

CREATE TABLE :

Creating a basic table involves naming the table and defining its columns and each column's datatypes.

SYNTAX :

```
CREATE TABLE  Table_name (
     Column1 datatype,
     column2 datatype,
     Column3 datatype
     ..........      );
```

EXAMPLE :

```
CREATE TABLE Student (
          Sid int,
          Name varchar(50),
          Address varchar(100),
               );
```

DROP TABLE :

The SQL DROP TABLE statementis used to delete the whole table from schema .

SYNTAX :

```
DROP TABLE  table_name;
```

EXAMPLE :

```
DROP TABLE   Employee;
```

TRUNCATE TABLE :

The TRUNCATE TABLE statement is used to delete only the data inside the table , but not the table itself.

SYNTAX :

```
TRUNCATE TABLE  table_name;
```

EXAMPLE :

```
TRUNCATE TABLE   Persons;
```

DDL COMMANDS - DATA DEFINITION LANGUAGE

ALTER TABLE :

A ALTER TABLE statement is used to add , delete or modify columns in an existing table.

- **ALTER TABLE - ADD COLUMN -** To add a column in a table.

SYNTAX :

```
ALTER TABLE  Table_name
ADD  Column_name datatype;
```

EXAMPLE :

```
ALTER TABLE  Student
ADD  address varchar(100);
```

- **ALTER TABLE - DROP COLUMN -** To delete a column in a table.

SYNTAX :

```
ALTER TABLE  Table_name
DROP COLUMN  Column_name ;
```

EXAMPLE :

```
ALTER TABLE  employees
DROP COLUMN  Email ;
```

- **ALTER TABLE - MODIFY COLUMN -** To modify/change column size or datatype in a table.

SYNTAX :

```
ALTER TABLE  Table_name
MODIFY  Column_name newDatatype;
```

EXAMPLE :

```
ALTER TABLE  Employee
MODIFY  Address varchar(200);
```

DDL COMMANDS - DATA DEFINITION LANGUAGE

RENAME TABLE :

A RENAME Statement is used to rename a table or a column in a table.

- **ALTER TABLE - RENAME COLUMN -** To rename a column in a table.

SYNTAX :

```
ALTER TABLE Table_name
RENAME COLUMN
OldColumnName to NewColumnName ;
```

EXAMPLE :

```
ALTER TABLE Student
RENAME COLUMN
Marks to average;
```

- **ALTER TABLE - RENAME TABLE -** To rename a Table in a schema.

SYNTAX :

```
ALTER TABLE Table_name
RENAME
OldTableName to NewTableName ;
```

EXAMPLE :

```
ALTER TABLE Student
RENAME Student to School ;
```

DML & DQL COMMANDS

DATA MANIPULATION LANGUAGE & DATA QUERY LANGUAGE

INSERT TABLE :

A INSERT INTO TABLE statement is used to add Values to the column in an existing table.

- **INSERT TABLE - ADD A SINGLE COLUMN -** To add a single value (V) set to the column(C) in a table.

SYNTAX :

```
INSERT INTO TABLE  Table_name
(C1,C2,C3)   VALUES    (V1,V2,V3) ;
```

EXAMPLE :

```
INSERT INTO TABLE   Student
(Id,name,place) VALUES  (1,'sri','chennai') ;
```

- **INSERT TABLE - ADD A MULTIPLE COLUMN -** To add a multiple value (V) sets to the column(C) in a table.

SYNTAX :

```
INSERT INTO TABLE   Table_name
(C1,C2,C3) VALUES
(V1,V2,V3) ,  (V1,V2,V3) ....;
```

EXAMPLE :

```
INSERT INTO TABLE   Student
(Id,name,place) VALUES
(1,'sri','chennai') ,(2,'ram','pune');
```

DML & DQL COMMANDS

DATA MANIPULATION LANGUAGE & DATA QUERY LANGUAGE

UPDATE AND DELETE TABLE :

A UPDATE Statement is used to update a values in a existing table and DELETE Statement is used to delete a particular row from the table

- **UPDATE TABLE - WITH WHERE CLAUSE :** Updating a particular row using where clause.

SYNTAX :

```
UPDATE   Table_name   SET
ColumnName = Value
WHERE  Condition;
```

EXAMPLE :

```
UPDATE  Student  SET
Dept = 'computer science'
WHERE  id=234 ;
```

- **UPDATE TABLE - HAVING MORE THAN ONE CONDITION :**

SYNTAX :

```
UPDATE   Table_name   SET
C1= V1,C2= V2
WHERE  Condition;
```

EXAMPLE :

```
UPDATE  Student  SET
Dept='EEE',name='Mani'
WHERE   Id=677;
```

- **DELETE TABLE - WITH WHERE CONDITION:** Deletes a particular row or a value from the table.

SYNTAX :

```
DELETE FROM   Table_name
WHERE  Condition;
```

EXAMPLE :

```
DELETE FROM  Employee
WHERE  s_id=999;
```

DML & DQL COMMANDS

DATA MANIPULATION LANGUAGE & DATA QUERY LANGUAGE

SELECT TABLE :

A SELECT Statement select the values in a existing table

- **SELECT TABLE - USING (*) OPERATOR:** Used to retrive all the datas from the table.

SYNTAX :

```
SELECT *  FROM   Table_name ;
```

EXAMPLE :

```
SELECT *  FROM    employee ;
```

- **SELECT TABLE - PARTICULAR DATA:**

SYNTAX :

```
SELECT  C1 , C2  FROM   Table_name ;
```

EXAMPLE :

```
SELECT   Name,age  FROM   Employee;
```

- **SELECT TABLE - USING WHERE CONDITION:**

SYNTAX :

```
SELECT   C1 , C2   Table_name
WHERE  Condition;
```

EXAMPLE :

```
SELECT  Dept,jobid   Employee
 WHERE  s_id=999;
```

SQL CONSTAINTS

PRIMARY KEY:

A PRIMARY KEY Constraint uniquely identifies each record in table.

- **CREATE :**

SYNTAX :

```
CREATE   TABLE   Table_name
            Colum1 datatype ,
            colum2 datatype,
            ...
            PRIMARY KEY(Column1) );
```

EXAMPLE :

```
CREATE   TABLE employee
            Id int,
            name varchar(100),
            Address varchar(100)
            PRIMARY KEY(id));
```

- **ALTER -ADD:**

SYNTAX :

```
ALTER TABLE  Table_name
ADD PRIMARY KEY (C1) ;
```

EXAMPLE :

```
ALTER TABLE  employee
ADD PRIMARY KEY (ID) ;
```

- **ALTER -DROP:**

SYNTAX :

```
ALTER TABLE   Table_name
DROP PRIMARY KEY (C1) ;
```

EXAMPLE :

```
ALTER TABLE  employee
DROP PRIMARY KEY (ID) ;
```

SQL CONSTAINTS

UNIQUE KEY:

A UNIQUE KEY Constaint does not allow duplicate values in acolum.

- **CREATE :**

SYNTAX :

```
CREATE   TABLE   Table_name
          Colum1 datatype ,
          colum2 datatype,
          ...
          UNIQUE (Column1) );
```

EXAMPLE :

```
CREATE   TABLE employee
          Id int,
          name varchar(100),
          Address varchar(100)
          UNIQUE(id));
```

- **ALTER -ADD:**

SYNTAX :

```
ALTER TABLE  Table_name
ADD UNIQUE (C1) ;
```

EXAMPLE :

```
ALTER TABLE  employee
ADD UNIQUE (ID) ;
```

- **ALTER -DROP:**

SYNTAX :

```
ALTER TABLE   Table_name
DROP INDEX  Column_name ;
```

EXAMPLE :

```
ALTER TABLE  employee
DROP INDEX Email;
```

SQL CONSTAINTS

CHECK (CONDITION) :

A CHECK Constraint is used to limit the range or allows a certain values for the column.

- **CREATE :**

SYNTAX :

```
CREATE   TABLE   Table_name
                 Colum1 datatype ,
                 colum2 datatype,
                 ...
                 CHECK(Condition ) );
```

EXAMPLE :

```
CREATE   TABLE employee
                Id int,
                name varchar(100),
                Address varchar(100)
                CHECK(age>18));
```

- **ALTER -ADD:**

SYNTAX :

```
ALTER TABLE  Table_name
ADD CHECK (CONDITION) ;
```

EXAMPLE :

```
ALTER TABLE  employee
ADD CHECK (AGE >18) ;
```

- **ALTER -DROP:**

SYNTAX :

```
ALTER TABLE   Table_name
DROP CHECK tablename_chk_1;
```

EXAMPLE :

```
ALTER TABLE  employee
DROP CHECK   employee_chk_1;
```

SQL CONSTAINTS

NOT NULL :

A NOT NULL constraint should contain values in the row. Does not allow NULLvalues.

- **CREATE :**

SYNTAX :

```
CREATE   TABLE   Table_name
            Colum1 datatype  NOTNULL,
            colum2 datatype,
            .....);
```

EXAMPLE :

```
CREATE   TABLE employee
            Id int,
            name varchar(100),
            Address varchar(100) NOTNULL,
            CHECK(age>18));
```

- **ALTER -ADD(MODIFY):**

SYNTAX :

```
ALTER TABLE  Table_name
MODIFY Column_name datatype NOT NULL ;
```

EXAMPLE :

```
ALTER TABLE  employee
MODIFY   age int   NOT NULL ;
```

- **ALTER -DROP(MODIFY):**

SYNTAX :

```
ALTER TABLE   Table_name
MODIFY   Column_name datatype  ;
```

EXAMPLE :

```
ALTER TABLE  employee
MODIFY  Column_name datatype  ;
```

SQL CONSTAINTS

Ra

DEFAULT :

A DEFAULT constraint used to set a default value to the column.

- **CREATE :**

SYNTAX :

```
CREATE   TABLE   Table_name
         Colum1 datatype  NOTNULL,
         colum2 datatype DEFAULT 'value',
         .....);
```

EXAMPLE :

```
CREATE   TABLE employee
         Id int,
         name varchar(100),
         Address varchar(100) NOTNULL,
          pincode int DEFAULT '8900654');
```

- **ALTER -ADD:**

SYNTAX :

```
ALTER TABLE  Table_name
ALTER COLUMN Column_name
SET DEFAULT  'Value' ;
```

EXAMPLE :

```
ALTER TABLE  employee
ALTER COLUMN  Place
SET DEFAULT 'Chennai' ;
```

- **ALTER -DROP:**

SYNTAX :

```
ALTER TABLE   Table_name
ALTER COLUMN   Column_name
DROP DEFAULT;
```

EXAMPLE :

```
ALTER TABLE  employee
ALTER COLUMN   Email
DROP DEFAULT;
```

SQL CONSTAINTS

Ra

FOREIGN KEY:

A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table.

- **CREATE :**

SYNTAX :

```
CREATE   TABLE   Table_name
            Colum1 datatype ,
            colum2 datatype,
            Foreign key(cname) references
            parent_tname(cname);
```

EXAMPLE :

```
CREATE   TABLE employee
            Id int,
            name varchar(100),
            Address varchar(100),
            Foreign key(id) references sales(sid);
```

- **ALTER -ADD:**

SYNTAX :

```
ALTER TABLE  Table_name
ADD FOREIGN KEY (Cname)
REFERENCES   Parent_t_name(cname);
```

EXAMPLE :

```
ALTER TABLE  employee
ADD FOREIGN KEY (Id)
REFERENCES  Employee_d(eid);
```

- **ALTER -DROP:**

SYNTAX :

```
ALTER TABLE   Table_name
DROP FOREIGN KEY   Tname_ibfk_1 ;
```

EXAMPLE :

```
ALTER TABLE  employee
DROP FOREIGN KEY    Stu_ibfk_1 ;
```

SQL OPERATORS

RANGE OPERATORS: BETWEEN , IN

Range operators are used to retrieve the data using the condition specified in ranges

- **BETWEEN OPERATOR :**

SYNTAX :

```
SELECT Column1,Column2
FROM Table_name
WHERE Column_name BETWEEN
value1 AND value 2;
```

EXAMPLE :

```
SELECT name,salary
FROM employee
WHERE salary BETWEEN
25000 AND 500000;
```

- **NOT BETWEEN OPERATOR :**

SYNTAX :

```
SELECT Column1,Column2
FROM Table_name
WHERE Column_name NOT BETWEEN
value1 AND value 2;
```

EXAMPLE :

```
SELECT name,salary
FROM employee
WHERE salary NOT BETWEEN
25000 AND 500000;
```

- **OR OPERATOR:**

SYNTAX :

```
SELECT * FROM Table_name
WHERE Condition1 OR Condition2;
```

EXAMPLE :

```
SELECT * FROM Employees
WHERE Id=09 OR place='chennai';
```

SQL OPERATORS

RANGE OPERATORS:

This operater used to retrive a particular range values from thre existing table.

- **IN OPERATOR :**

SYNTAX :

```
SELECT * FROM  Table_name
 WHERE   Column_name  IN (VALUES);
```

EXAMPLE :

```
SELECT * FROM  student;
 WHERE   Age   IN (23,26);
```

- **NOT IN OPERATOR :**

SYNTAX :

```
SELECT * FROM  Table_name
 WHERE   Column_name  NOT IN (VALUES);
```

EXAMPLE :

```
SELECT * FROM  student;
WHERE   Age   NOT IN (23,26);
```

- **WILDCARD:** SQL WILDCARD is a special characters

LIKE OPERATORS- TYPES OF LIKE OPERATORS

- **%** SQL WILDCARD is a special characters
- **_** SQL WILDCARD is a special characters
- **[ ]** SQL WILDCARD is a special characters

SQL OPERATORS

LIKE OPERATORS:

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

- **% OPERATOR :**

SYNTAX :

```
SELECT * FROM  Table_name
 WHERE   Column_name  LIKE  '% VALUE ';
                      LIKE  ' VALUE % ';
                      LIKE  ' %VALUE % ';
```

EXAMPLE :

```
SELECT * FROM  customer
 WHERE   C_name  LIKE  '% A';
                 LIKE  ' Y % ';
                 LIKE  ' %T % ';
```

- **__ OPERATOR :**

SYNTAX :

```
SELECT * FROM  Table_name
 WHERE   Column_name  LIKE  '_ VALUE %';
                       LIKE  ' __VALUE%  ';
```

EXAMPLE :

```
SELECT * FROM  customer
 WHERE   C_name LIKE  '_  %';
                LIKE  ' __E%  ';
```

SQL OPERATORS

ORDER BY:

The ORDER BY command is used to sort the result set in ascending or descending order

- **ASCENDING AND DESCENDING:**

SYNTAX :

```
SELECT * FROM  Table_name
ORDER BY  Column_name  ASC/DESC ;
```

EXAMPLE :

```
SELECT * FROM  customer
ORDER BY   Age   ASC/DESC ;
```

- **ASCENDING AND DESCENDING- WITH WHERE CONDITION-**

SYNTAX :

```
SELECT * FROM  Table_name
 WHERE   Condition
ORDER BY  Column_name  ASC/DESC ;
```

EXAMPLE :

```
SELECT * FROM  customer
 WHERE   Age > 18
ORDER BY  Salary   ASC/DESC ;
```

- **LIMIT AND OFFSET-** .

SYNTAX :

```
SELECT * FROM  Table_name
ORDER BY Column_name  ASC/DESC ;
LIMIT     OFFSET ;
```

EXAMPLE :

```
SELECT * FROM  customer
ORDER BY Salary  ASC/DESC ;
LIMIT 1   OFFSET 2;
```

SQL OPERATORS

- **DISTINCT :** The distinct keyword is used in conjunction with the select keyword.

SYNTAX :

```
SELECT DISTINCT  Column_name
FROM  Table_name ;
```

EXAMPLE :

```
SELECT DISTINCT  Name
FROM  Employees ;
```

- **IS NULL / IS NOTNULL -** In SQL, IS NULL and IS NOT NULL are used to check if a column in a table contains a NULL value or not.

SYNTAX :

```
SELECT * FROM  Table_name
 WHERE  Column_name  IS NULL / IS NOT NULL;
```

EXAMPLE :

```
SELECT * FROM  customer
 WHERE  Address  IS NULL / IS NOT NULL;
```

- **CASE EXPRESSION:** returns a value for the condition specified.

SYNTAX :

```
 SELECT  Column1,colum
 CASE  WHEN  condition1  THEN  'result1',
       WHEN condition1  THEN  'result2',
       ELSE  'result3'
END AS  Temp_column_name
FROM  Table_name ;
```

EXAMPLE :

```
 SELECT  Name,salary
 CASE  WHEN  Salary>12000   THEN  'Greater',
       WHEN  Salary <12000  THEN  'Lowest',
       ELSE  'Equal'
END AS  Result
FROM  Employees ;
```

SQL FUNCTIONS

STRING FUNCTION:

- **CONCAT():** Adds the strings

QUERY :
```
SELECT CONCAT( 'Good' ,'Morning');
```
OUTPUT :
```
GoodMorning
```

- **LOWER():** Prints lower value

QUERY :
```
SELECT LOWER('MYSQLQUERY');
```
OUTPUT :
```
mysqlquery
```

- **UPPER():** Prints the upper value

QUERY :
```
SELECT UPPER    ('priya');
```
OUTPUT :
```
PRIYA
```

- **SUBSTRING():** Prints the mentioned string

QUERY :
```
SELECT SUBSTR   (Eventually',2,3);
```
OUTPUT :
```
ven
```

- **REPLACE():** Replaces the string

QUERY :
```
SELECT REPLACE('Good' ,'Morning','Good','Pleasent');
```
OUTPUT :
```
Pleasent Morning
```

- **LENGTH ():** Prints the length of the string

QUERY :
```
SELECT LENGTH(  'I play football');
```
OUTPUT :
```
13
```

SQL FUNCTIONS

MATH FUNCTION:

- **ABSOLUTE():** Converts the negative value to positive

QUERY :	**OUTPUT :**
SELECT ABS(-588);	588

- **CEILING():** Prints the next whole number

QUERY :	**OUTPUT :**
SELECT CEIL(35.7);	36

- **FLOOR():** Prints the previous whole number

QUERY :	**OUTPUT :**
SELECT FLOOR (35.6);	35

- **ROUND():** Rounds the decimal values

QUERY :	**OUTPUT :**
SELECT ROUND (39.877,2);	39.88

- **MODULES():** Prints the reminder

QUERY :	**OUTPUT :**
SELECT MOD(12,2);	0

- **TRUNCATE ():** Drops the values after decimal

QUERY :	**OUTPUT :**
SELECT TRUNCATE(37.7890,3);	337.789

SQL FUNCTIONS

DATE FUNCTION:

- **CURRENT DATE():** Prints the current date

QUERY :

```
SELECT CUR DATE();
```

OUTPUT :

```
2023-08-22
```

- **NOW():** Prints the current date

QUERY :

```
SELECT NOW();
```

OUTPUT :

```
2023-08-22
```

- **SYSTEM DATE():** Prints the system date

QUERY :

```
SELECT SYSDATE();
```

OUTPUT :

- **MONTH(NOW()):**Prints the current month

QUERY :

```
SELECT MONTH(NOW());
```

OUTPUT :

```
08
```

SQL FUNCTIONS

AGGREGATE FUNCTION:

- **COUNT():** Counts the value from the given column.

SYNTAX
```
SELECT COUNT(CNAME) FROM tname;
```
EXAMPLE
```
SELECT COUNT(ID) FROM Student;
```

- **MIN():** Prints the minimum value from the given column.

SYNTAX
```
SELECT MIN(CNAME) FROM tname;
```
EXAMPLE
```
SELECT MIN(SALARY) FROM customer;
```

- **MAX():** Prints the max value from the given column.

SYNTAX
```
SELECT MAX(CNAME) FROM tname;
```
EXAMPLE
```
SELECT MAX(FEE) FROM Student;
```

- **AVERAGE():** Prints the average value from the given column.

SYNTAX
```
SELECT AVG(CNAME) FROM tname;
```
EXAMPLE
```
SELECT AVG(MARKS) FROM Student;
```

- **SUM():** Prints the sum of value from the given column.

SYNTAX
```
SELECT SUM(CNAME) FROM tname;
```
EXAMPLE
```
SELECT SUM(MARKS) FROM Student;
```

SQL FUNCTIONS

GROUP BY - HAVING CLAUSE

- Used by group the row that have the same value.
- WHERE clause cannot be used without group by.
- Implemented in row operation
- HAVING clause after GROUP BY

SYNTAX

```
SELECT   Column(s),aggregate_function(column)
FROM     Table_name
GROUP BY  Column_name
HAVING   condition ;
```

EXAMPLE

```
SELECT   Id,max(salary)
FROM     employee
GROUP BY  Id
HAVING   Id=4667 ;
```

SQL FUNCTIONS

SUBQUERY

- Sub query must enclosed with parenthesis.
- A subquery can have only one column in the main query.
- ORDER BY, GROUP BY AND BETWEEN operator cannot be used in subquery.

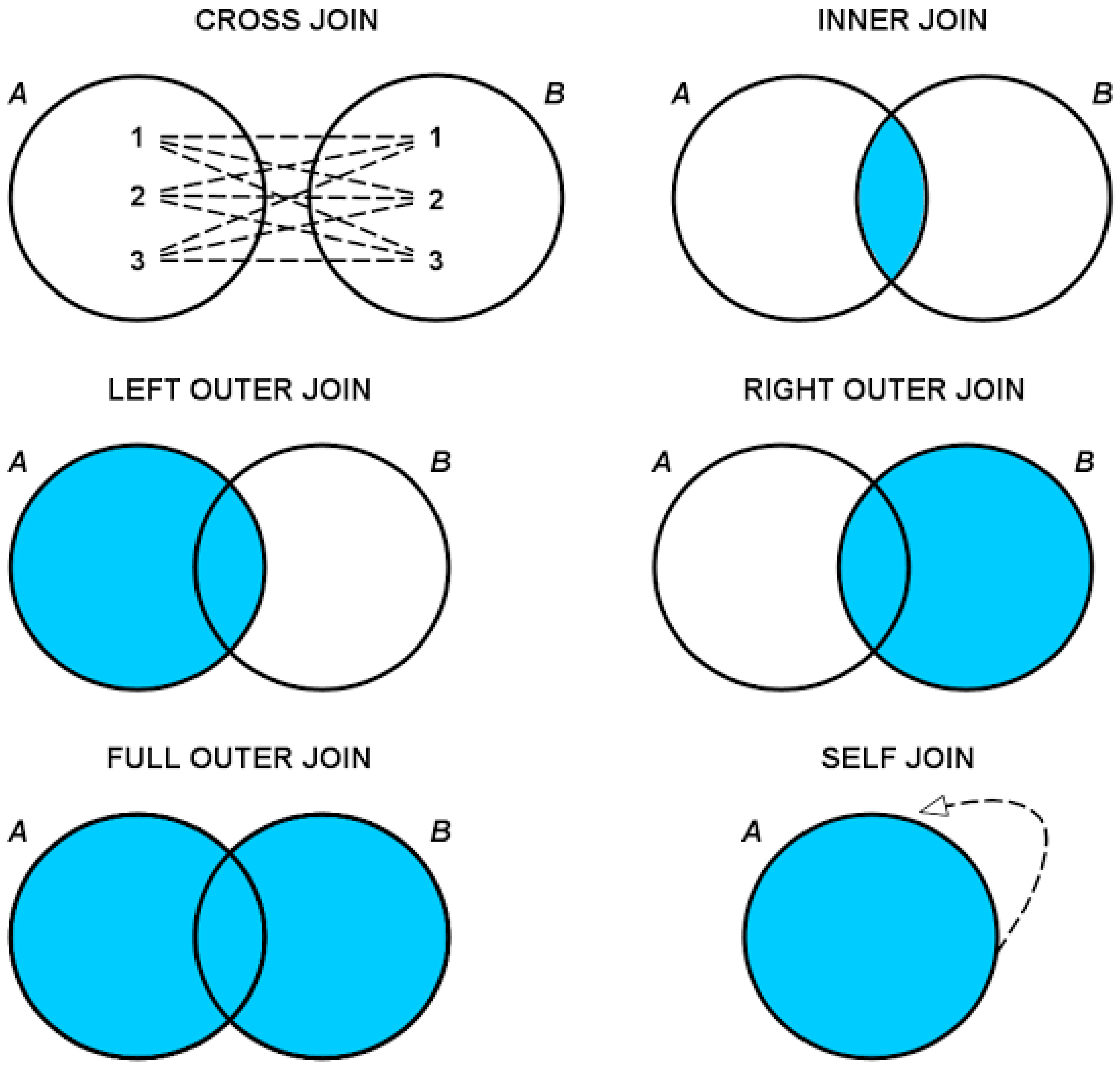
SYNTAX

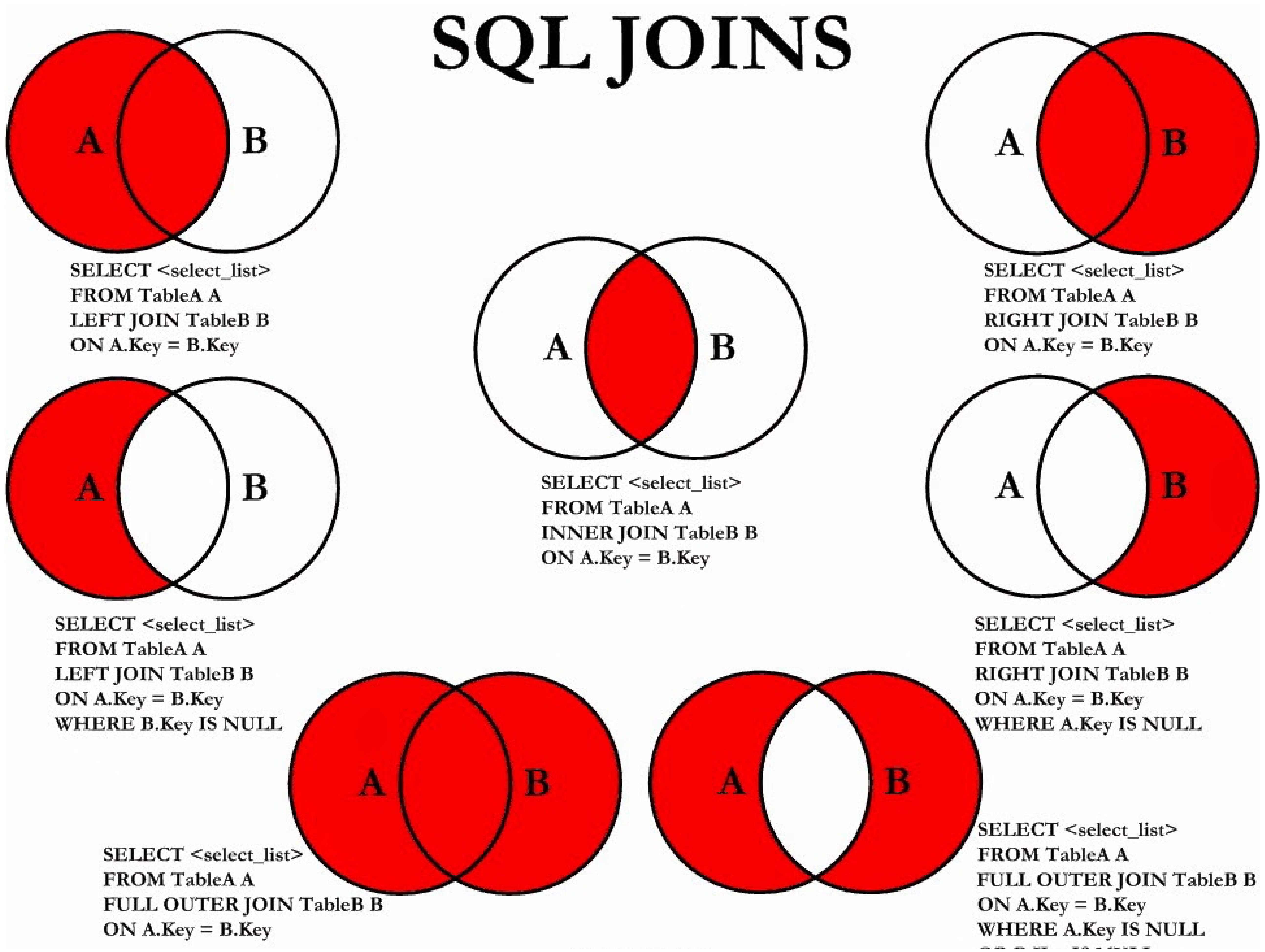
```
SELECT *  FROM Table_name
WHERE   Column_name  operator
   (SELECT  Column_name  FROM
    Table_name );
```

EXAMPLE

```
SELECT *  FROM Employees
WHERE   salary =
   (SELECT  Max(salary)  FROM
    Employees);
```

JOINS


CROSS JOIN
A
B
1
2
3
1
2
3
INNER JOIN
A
B
LEFT OUTER JOIN
A
B
RIGHT OUTER JOIN
A
B
FULL OUTER JOIN
A
B
SELF JOIN
A



SQL JOINS

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```

```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```