5 Types commands in Sql :-

1. DDL (Data Definition Language)

- DDL commands define, modify, and manage database objectlike table , schema, and indexes.

◆ **Examples :-**

- CREATE – creates a new table or database
- ALTER – modifies an existing table
- DROP – deletes a table or database
- TRUNCATE – removes all records from a table

2. DML (Data Manipulation Language)

- Dml commands handle data within tables, such as inserting, updating, or deleting rows.

◆ **Examples :-**

- INSERT – adds new data
- UPDATE – modifies existing data
- DELETE – removes data

3. DCL (Data Control Language)

- DCL commands control user access and permissions to the database objects.

◆ **Examples :-**

- GRANT – gives user access rights
- REVOKE – removes access rights

4. TCL (Transaction Control Language)

- TCL commands manage database transactions, ensuring data consistency and integrity.

◆ **Examples :-**

- COMMIT – saves the transaction
- ROLLBACK – undoes the transaction
- SAVEPOINT – sets a point within a transaction

5. DQL (Data Query Language)

- DQL focuses on querying data from the databases .

◆ **Example :-**

- SELECT – retrieves data from one or more tables

Create Database :-

- It is used to create database in sql. **Example :-** CREATE DATABASE company ;

Delete Database :-

- It is used to delete database in sql. **Example :-** DROP DATABASE company ;

Create Table :-

- It is used to create table in sql.

```sql
CREATE TABLE employee (
    employee_id SERIAL PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    position VARCHAR (50),
    department VARCHAR(50),
    hire_date DATE,
    age INT
);

SELECT * FROM employee;
```

Serial in Database :-

- Database create karte waqt serial likhne ka use primary key ya unique ID field ke liye hota hai. Yeh automatic number generate karta hai har new row ke liye.

Insert Data in Table :-

- It is used to insert data in table.

Example :-

```
INSERT INTO employee(name,position,department,hire_date,age)
VALUES ('jil','Data Analystics','Data Science','2022-05-15',65000),
       ('Priya','Software engineer','IT','2021-09-03',50000);

SELECT * FROM employee;
```

Output :-

	employee_id [PK] integer	name character varying (50)	position character varying (50)	department character varying (50)	hire_date date	age integer
1	1	jil	Data Analystics	Data Science	2022-05-15	65000
2	2	jil	Data Analystics	Data Science	2022-05-15	65000
3	3	Priya	Software engineer	IT	2021-09-03	50000

ALTER column :-

- In SQL (Structured Query Language), the ALTER statement is used to modify an existing table.

Add Multiple Columns

- **Syntax :-** ALTER TABLE table_name ADD (column1 datatype, column2 datatype) ;
- **Example :-** ALTER TABLE students ADD (age INT, dob DATE) ;

Modify Column Data Type

- **Syntax :-** ALTER TABLE table_name MODIFY column_name new_datatype ;
- **Example :-** ALTER TABLE students MODIFY age VARCHAR(3) ;

Change or Rename Column Name

- **Syntax :-** ALTER TABLE table_name CHANGE old_column_name new_column_name datatype ;
- **Example :-** ALTER TABLE students CHANGE age student_age INT ;

Drop a Column

- **Syntax :-** ALTER TABLE table_name DROP COLUMN column_name ; **Example :-** ALTER TABLE students DROP COLUMN age ;

Rename a Table

- **Example :-** RENAME TABLE students TO student_info ; or **Example :-** ALTER TABLE app_users RENAME TO customers;

Add NOT NULL constraint to city column

- **Example :-** ALTER TABLE users ALTER COLUMN city SET NOT NULL ;

Add Check constraint to age column

- **Example :-** ALTER TABLE users ALTER ADD CONSTRAINT age CHECK (age >= 18) ;

Truncate command :-

- The TRUNCATE command in SQL is used to delete all rows from a table quickly, without logging each row deletion like DELETE does.
- Syntax :- TRUNCATE TABLE table_name ;
- Example Truncate :- TRUNCATE TABLE employee RESTART IDENTITY ;

Note :- When we delete Row we can use delete command and when we delete column , table , database we can use Drop Command.

Delete Row and Column from Table :-

Delete Command

- The DELETE command is used to remove one or more rows from a table.
- **Syntax :-** DELETE FROM table_name WHERE condition;
- **Example Delete Row :-** DELETE FROM employee WHERE employee_id = 105 ;
- **Example Delete Column :-** ALTER TABLE employee2 DROP COLUMN salary ;
- **Example Delete Table :-** DROP TABLE employee 2 ;
- **Example Delete Database :-** DROP DATABASE IF EXISTS company ;

Data Types in sql :-

Numeric data types

Data Type	Description	Example Use
INTEGER	Store whole numbers	Employee IDs, Age
SERIAL	Auto-incrementing integer	Primary key auto-incement
NUMEIC(p , s)	Stores exact number with precision (p) and Scale (s)	Financial Data (Salary)
REAL	Store floating point numbers	Scientific calculation

Character Data Type

Data Type	Description	Example Use
CHAR(n)	Fixed length string of n character	Employees codes
VARCHAR(n)	Variable-length string up to n characters	Name, email, address
TEXT	Unlimited-length string	Description, comments

Date and Time Data Type

Data Type	Description	Example Use
DATE	Store date (year, month, day)	Birthdate, Hire date
TIME	Stores Time (hour, minute , second)	Appointment times
TIMESTAMP	Store date and time	Order timestamp
TIMESTAMPTZ	Store data and time with timezone info.	Global Event Tracking

Boolean Data Type :- For check user is active or not.

Data Type	Description	Example Use
BOOLEAN	Store True, False, or Null	Flags, Active status

Constraints in SQL :-

- SQL Constraints are rules applied on columns to ensure valid and accurate data is stored in the database.
- Constraints help maintain the accuracy, validity, and integrity of the data in the database.

Types of SQL Constraints :-

NOT NULL :-

- The NOT NULL constraint ensures that a column cannot have a NULL (empty) value.

Example :- CREATE TABLE employees (id INT, name VARCHAR(100) NOT NULL);

UNIQUE :-

- The UNIQUE constraint ensures that all values in a column are different from each other.

Example :-

CREATE TABLE students (id INT, email VARCHAR(100) UNIQUE);

INSERT INTO students (id, email) VALUES (1, 'a@example.com'); or **error if this :-** INSERT INTO students (id, email) VALUES (2, 'a@example.com');

PRIMARY KEY :-

- The PRIMARY KEY constraint uniquely identifies each record in a table.

Example :-

CREATE TABLE users (user_id INT PRIMARY KEY, username VARCHAR(100));

INSERT INTO users VALUES (1, 'zeel'); or **error if this :-** INSERT INTO users VALUES (1, 'admin');

CHECK :-

- The CHECK constraint is used to limit the value that can be placed in a column based on a condition.
- For example, a CHECK constraint can ensure that the "age" column must always be greater than or equal to 18.

Example :-

CREATE TABLE accounts (acc_id INT, balance INT CHECK (balance >= 0));

INSERT INTO accounts VALUES (1, 1000); or **error if this :-** INSERT INTO accounts VALUES (2, -200);

DEFAULT :-

- The DEFAULT constraint provides a default value for a column when no value is specified.

Example :-

CREATE TABLE orders (order_id INT, status VARCHAR(20) DEFAULT 'Pending');

INSERT INTO orders (order_id) VALUES (101);

AUTO_INCREMENT :-

- The AUTO_INCREMENT constraint is used to automatically generate a unique number when a new record is inserted.

Example :-

CREATE TABLE products (product_id INT AUTO_INCREMENT PRIMARY KEY, product_name VARCHAR(100));

INSERT INTO products (product_name) VALUES ('Book');

INSERT INTO products (product_name) VALUES ('Pen');

IF Exist or IF NoT EXIST :- In SQL, IF EXISTS and IF NOT EXISTS clauses are used to safely manage database objects like tables, databases, views, etc., to prevent errors when trying to create or delete something that already exists or doesn't exist.

Example IF NOT EXIST :- CREATE TABLE IF NOT EXISTS users (id INT PRIMARY KEY, name VARCHAR(100));

Example IF EXIST :- DROP TABLE IF EXISTS users;

Update Data in Sql :-

- The UPDATE statement is used to modify existing records in a table.
- **Syntax :-** UPDATE table_name SET column1 = value1, column2 = value2, ... WHERE condition;
- **Example :-** UPDATE students SET name = 'Zeel Patel', age = 22 WHERE id = 1;

ORDER BY in Sql :-

- ASCENDING and DESCENDING order are used with the ORDER BY clause to sort data when retrieving it from a table.
- **Syntax :-** SELECT * FROM table_name ORDER BY column_name [ASC | DESC] ;
- **Example :-** SELECT First_name, Last_name, salary FROM students ORDER BY salary ASC ;

Assignment :-

```
-- Drop the table if it already exists
DROP TABLE IF EXISTS employees;

-- Create the employees table
CREATE TABLE employees (
    employee_id SERIAL PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    department VARCHAR(50),
    salary DECIMAL(10, 2) CHECK (salary > 0),
    joining_date DATE NOT NULL,
    age INT CHECK (age >= 18)
);

SELECT * FROM employees;

-- Insert data into employees table
INSERT INTO employees (first_name, last_name, department, salary, joining_date, age) VALUES
('Amit', 'Sharma', 'IT', 60000.00, '2022-05-01', 29),
('Neha', 'Patel', 'HR', 55000.00, '2021-08-15', 32),
('Ravi', 'Kumar', 'Finance', 70000.00, '2020-03-10', 35),
('Anjali', 'Verma', 'IT', 65000.00, '2019-11-22', 28),
('Suresh', 'Reddy', 'Operations', 50000.00, '2023-01-10', 26);

-- Assignment Questions
--Q1 : Retrieve all employees' first_name and their departments.
SELECT FIRST_NAME, DEPARTMENT FROM EMPLOYEES;
```

--Q2 : Update the salary of all employees in the 'IT' department by increasing it by 10%.

```
UPDATE employees
SET salary=salary + (salary*0.1 )
WHERE department ='IT';
```

--Q3 : Delete all employees who are older than 34 years.

```
DELETE FROM employees WHERE age>34;
```

--Q4 : Add a new column `email` to the `employees` table.

```
ALTER TABLE employees ADD COLUMN email VARCHAR(100);
```

--Q5 : Rename the `department` column to `dept_name`.

```
ALTER TABLE employees RENAME COLUMN department TO dept_name;
```

--Q6 : Retrieve the names of employees who joined after January 1, 2021.

```
SELECT first_name, last_name, joining_date FROM employees WHERE joining_date > '2021-01-01';
```

--Q7 : List all employees with their age and salary in descending order of salary.

```
SELECT first_name, age, salary FROM employees  ORDER BY salary DESC;
```

--Q8 : Insert a new employee with the following details : -- ('Raj', 'Singh', 'Marketing', 60000, '2023-09-15', 30).

```
INSERT INTO employees(first_name, last_name, dept_name, salary, joining_date, age) VALUES('Raj', 'Singh', 'Marketing', 60000, '2023-09-15', 30);
```

--Q9 : Update age of employee +1 to every employee.

```
UPDATE employees SET age=age+1;
```

Arithmetic Operator :-

Operator	Description	Example
+	Addition	salary + bonus
-	Subtraction	price - discount
*	Multiplication	quantity * rate
/	Division	total / count
%	Modulus (remainder)	marks % 5

Addition :- SELECT salary, bonus, salary + bonus AS total_salary FROM employees;

Subtraction :- SELECT price, discount, price - discount AS final_price FROM products;

Multiplication :- SELECT quantity, rate, quantity * rate AS total_amount FROM orders;

Division :- SELECT total, count, total / count AS average FROM sales;

1. Retrieve the First_name, salary, and calculate a 10% bonus on salary

```
SELECT first_name, salary, (salary + 0.10 ) As Bonus From employee2
```

Comparison Operators :-

Operator	Description	Example
=	Equal to	age = 25
!= or <>	Not equal to	salary != 50000
>	Greater than	marks > 40
<	Less than	marks < 90
>=	Greater than or equal to	age >= 18
<=	Less than or equal to	experience <= 5
BETWEEN	Within a range	marks BETWEEN 60 AND 80
LIKE	Pattern match	name LIKE 'Z%'
IN	Matches any in a list	city IN ('Surat', 'Rajkot')
IS NULL	Value is NULL	email IS NULL

Equal to (=) :- SELECT * FROM students WHERE city = 'Surat';

Not equal to (!=) :- SELECT first_name, age From employee2 WHERE age!=30;

Greater than (>) :- SELECT * FROM students WHERE age > 20;

Less than (<) :- SELECT * FROM students WHERE marks < 60;

Greater than or equal (>=) :- SELECT * FROM students WHERE marks >= 75;

Between :- SELECT * FROM students WHERE marks BETWEEN 50 AND 90;

IN :- SELECT * FROM students WHERE city IN ('Surat', 'Rajkot');

IS NULL :- SELECT * FROM students WHERE city IS NULL;

Logical Operators :-

Operator	Description	Example
AND	Returns TRUE if both conditions are true	age > 18 AND city = 'Surat'
OR	Returns TRUE if any one condition is true	age < 18 OR marks > 75
NOT	Reverses the result (TRUE ⟶ FALSE)	NOT (city = 'Surat')

AND ⟶ Both conditions must be true :- SELECT * FROM students WHERE age > 18 AND marks > 60 ;

OR ⟶ At least one condition must be true :- SELECT * FROM students WHERE age < 18 OR marks < 60;

NOT ⟶ Reverses the condition :- SELECT * FROM students WHERE NOT (city = 'Surat');

Combine AND and OR (with parentheses) :- SELECT * FROM students WHERE (marks > 70 AND age > 18) OR city = 'Mumbai';

Limit in Sql :- LIMIT is used to restrict the number of rows returned by a query.

- It is especially useful when you want to show only a few records, such as the top 5 students, latest 10 orders, etc.
- **Syntax :-** SELECT column1, column2, ... FROM table_name LIMIT number;
- **Return first 3 records :-** SELECT * FROM students LIMIT 3;
- **Return Last 2 Students :-** SELECT * FROM students ORDER BY id DESC LIMIT 2;

Set Operators in SQL :-

- Set Operators are used to combine the results of two or more SELECT queries.

Operator	Description
UNION	Combines results of both queries, removes duplicates
UNION ALL	Combines results of both queries, includes duplicates
INTERSECT	Returns only common rows from both queries
EXCEPT	Returns rows from first query that don't exist in second

Example :-

```
DROP TABLE IF EXISTS students_2023;
CREATE TABLE students_2023 ( student_id INT PRIMARY KEY, student_name VARCHAR(100), course VARCHAR(50) );
INSERT INTO students_2023 (student_id, student_name, course) VALUES
(1, 'Aarav Sharma', 'Computer Science'),
(2, 'Ishita Verma', 'Mechanical Engineering'),
(3, 'Kabir Patel', 'Electronics'),
(4, 'Ananya Desai', 'Civil Engineering'),
(5, 'Rahul Gupta', 'Computer Science');
SELECT * FROM students_2023;

DROP TABLE IF EXISTS students_2024;
CREATE TABLE students_2024 ( student_id INT PRIMARY KEY, student_name VARCHAR(100), course VARCHAR(50) );
INSERT INTO students_2024 (student_id, student_name, course) VALUES
(3, 'Kabir Patel', 'Electronics'),   -- Same as students_2023
(4, 'Ananya Desai', 'Civil Engineering'), -- Same as students_2023
(6, 'Meera Rao', 'Computer Science'),
(7, 'Vikram Singh', 'Mathematics'),
(8, 'Sanya Kapoor', 'Physics');
SELECT * FROM students_2024;

-- UNION -- Combines results, removes duplicates
SELECT student_name, course FROM students_2023
UNION
SELECT student_name, course FROM students_2024;

-- UNION ALL - Combines results, keeps duplicates
SELECT student_name, course  FROM students_2023
UNION ALL
SELECT student_name, course FROM students_2024;
```

-- INTERSECT - Returns common results in both tables

```
SELECT student_name, course  FROM students_2023
INTERSECT
SELECT student_name, course FROM students_2024;
```

-- EXCEPT -- Returns results in the first table avaliable but not in the second

```
SELECT student_name, course  FROM students_2023
EXCEPT
SELECT student_name, course FROM students_2024;
```

Functions in SQL :-

- Functions are built-in operations that allow you to perform calculations, transform data, or return useful information from the database.

Aggregate Functions :- Aggregate functions perform calculations on multiple rows of a table and return a single value.

Function	Description	Example
COUNT( )	Counts number of rows	SELECT COUNT(*) FROM students;
SUM( )	Adds all values	SELECT SUM(salary) FROM emp;
AVG( )	Calculates average	SELECT AVG(marks) FROM students;
MIN( )	Finds minimum value	SELECT MIN(age) FROM students;
MAX( )	Finds maximum value	SELECT MAX(salary) FROM emp;

Example :-

```
DROP TABLE IF EXISTS products;
CREATE TABLE products (  product_id SERIAL PRIMARY KEY,  product_name VARCHAR(100),  category VARCHAR(50), price NUMERIC(10, 2),
quantity INT,  added_date DATE, discount_rate NUMERIC(5, 2) );

INSERT INTO products (product_name, category, price, quantity, added_date, discount_rate) VALUES
('Laptop', 'Electronics', 75000.50, 10, '2024-01-15', 10.00),
('Smartphone', 'Electronics', 45000.99, 25, '2024-02-20', 5.00),
('Headphones', 'Accessories', 1500.75, 50, '2024-03-05', 15.00),
('Office Chair', 'Furniture', 5500.00, 20, '2023-12-01', 20.00),
('Desk', 'Furniture', 8000.00, 15, '2023-11-20', 12.00),
('Monitor', 'Electronics', 12000.00, 8, '2024-01-10', 8.00),
('Printer', 'Electronics', 9500.50, 5, '2024-02-01', 7.50),
('Mouse', 'Accessories', 750.00, 40, '2024-03-18', 10.00),
('Keyboard', 'Accessories', 1250.00, 35, '2024-03-18', 10.00),
('Tablet', 'Electronics', 30000.00, 12, '2024-02-28', 5.00);

SELECT * FROM products;
```

```
-- Total Quantity Available of all products

SELECT SUM(quantity) AS total_quantity   FROM products;

-- sum with condition

SELECT SUM(quantity) AS quantity_of_ele    FROM products     WHERE category='Electronics' AND price > 20000 ;

-- Total number of products

SELECT COUNT(*) AS total_products   FROM products;

-- count with condition

SELECT COUNT(*) AS total_products   FROM products   WHERE product_name LIKE '%phone%';

-- Average Price of Products

SELECT AVG(price) AS average_price  FROM products;

-- Average Price of Products with condition

SELECT AVG(price) AS average_price  FROM products

WHERE category='Accessories' OR added_date > '2024-02-01';

SELECT * FROM products;

-- Maximum and Minimum price

SELECT MAX(price) AS MAX_PRICE,  MIN(PRICE) AS MIN_PRICE    FROM products;
```

String Functions :- String functions are used to manipulate or analyze text data.

Function	Description	Example
LENGTH(str)	Length of string	SELECT LENGTH('Zeel'); ⟶ 4
LOWER(str)	Convert to lowercase	SELECT LOWER('HELLO'); ⟶ hello
UPPER(str)	Convert to uppercase	SELECT UPPER('hello'); ⟶ HELLO
CONCAT(s1, s2)	Join strings	SELECT CONCAT('Zeel', ' Patel');
SUBSTRING(str, start, len)	Get part of string	SELECT SUBSTRING('ZeelPatel', 1, 4); ⟶ Zeel

```
SELECT * from products;

-- Get all the categories in Uppercase

SELECT UPPER(category) AS Catergory_Capital  FROM products;

-- Get all the categories in Lowercase

SELECT LOWER(category) AS Catergory_Capital  FROM products;

-- Join Product_name adn category text with hypen.

SELECT CONCAT(product_name,'-',category) As product_details   FROM products;

-- Extract the first 5 characters from product_name

SELECT SUBSTRING(product_name, 1,5) AS short_name  FROM products;

-- Count length

SELECT product_name, LENGTH(product_name) AS COUNT_OF_CHAR  FROM products;

-- Replace the word "phone" with "device" in product names

SELECT REPLACE(product_name, 'phone','device') AS updated  FROM products;
```

Date/Time Functions :- Date/Time functions are used to work with date and time values in a database.

Function	Description	Example
NOW( )	Current date and time	SELECT NOW();
CURRENT_DATE( )	Current date	SELECT CURRENT_DATE();
CURRENT_TIME( )	Current time	SELECT CURRENT_TIME();
YEAR(date)	Extract year	SELECT YEAR('2023-08-01'); ⟶ 2023
EXTRACT( )	Extract	Extract specific part of date

```
SELECT * from products;

-- 1. NOW( ) – Get Current Date and Time

SELECT NOW( ) AS Current_Datetime;

-- 2. EXTRACT( ) – Extract Parts of a Date   -- Extract the year, month, and day from the added_date column.

SELECT product_name,
       EXTRACT(YEAR FROM added_date) AS Year_Added,
       EXTRACT(MONTH FROM added_date) AS Month_Added,
       EXTRACT(DAY FROM added_date) AS Day_Added
FROM products;

-- 3. AGE( ) – Calculate Age Between Dates -- Calculate the time difference between added_date and today's date.

SELECT product_name, AGE(CURRENT_DATE, added_date) AS Age_since_added  FROM products;

-- 4. CURRENT_TIME( ) – Get Current Time  --Retrieve only the current time.

SELECT CURRENT_TIME AS current_time;

-- 5. CURRENT_DATE( ) – Get Current Date

SELECT CURRENT_DATE AS today_date;
```

Conditional Function :- Conditional functions in SQL return different values based on conditions, similar to if-else logic in programming.

```
SELECT * from products;

/* 1. CASE Function – Categorizing Based on Conditions

 We will categorize products into price ranges :

Expensive if the price is greater than or equal to 50,000.

Moderate if the price is between 10,000 and 49,999.

Affordable if the price is less than 10,000.

*/

SELECT product_name, price,
       CASE
          WHEN price>=50000 THEN 'Expensive'
          WHEN price>=10000 AND price<=49999 THEN 'Moderate'
          ELSE 'Affordable'
       END AS price_category
FROM products;
```

Assignment on Case Conversion :-

```
SELECT * from products;

/* 1. CASE Function – Categorizing Based on Conditions

 We will categorize products into price ranges :-

Expensive if the price is greater than or equal to 50,000.

Moderate if the price is between 10,000 and 49,999.

Affordable if the price is less than 10,000. */

SELECT product_name, price,
	CASE
	   WHEN price>=50000 THEN 'Expensive'
	   WHEN price>=10000 AND price<=49999 THEN 'Moderate'
	   ELSE 'Affordable'
	END AS price_category
FROM products;


/* 2. CASE with AND & OR Operators – Stock Status

We will classify products based on quantity available :-

In Stock if quantity is 10 or more.

Limited Stock if quantity is between 5 and 9.

Out of Stock Soon if quantity is less than 5.

*/

SELECT product_name, quantity,
	CASE
	   WHEN quantity >=10 THEN 'InStock'
	   WHEN quantity BETWEEN 6 AND 9 THEN 'Limited stock'
	   ELSE 'Out of stock soon'
	END AS stock_status
FROM products;


/* 3. CASE with LIKE Operator – Category Classification

Check if the category name contains "Electronics" or "Furniture" using LIKE. */

SELECT product_name, category,
	CASE
	   WHEN category LIKE 'Electronics%' THEN 'Electronic Item'
	   WHEN category LIKE 'Furniture%' THEN 'Furniture Item'
	   ELSE 'Accessory Item'
	END AS category_Status
FROM products;
```

Joins In Sql :-

- A JOIN in SQL is used to combine rows from two or more tables based on a related column.
- It helps retrieve data spread across multiple tables.

Table Create :-

```
-- Create Employees Table
CREATE TABLE Employees3 (
    employee_id SERIAL PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    department_id INT
);


-- Insert Data into Employees
INSERT INTO Employees3 (first_name, last_name, department_id)
VALUES
('Rahul', 'Sharma', 101),
('Priya', 'Mehta', 102),
('Ankit', 'Verma', 103),
('Simran', 'Kaur', NULL),
('Aman', 'Singh', 101);

SELECT * FROM employees3;
```

	employee_id [PK] integer	first_name character varying (50)	last_name character varying (50)	department_id integer
1	1	Rahul	Sharma	101
2	2	Priya	Mehta	102
3	3	Ankit	Verma	103
4	4	Simran	Kaur	[null]
5	5	Aman	Singh	101

```
-- Create Departments Table
CREATE TABLE Departments (
    department_id INT PRIMARY KEY,
    department_name VARCHAR(50)
);


-- Insert Data into Departments
INSERT INTO Departments (department_id, department_name)
VALUES
(101, 'Sales'),
(102, 'Marketing'),
(103, 'IT'),
(104, 'HR');

SELECT * FROM Departments;
```

	department_id [PK] integer	department_name character varying (50)
1	101	Sales
2	102	Marketing
3	103	IT
4	104	HR

1. INNER JOIN :-

- The INNER JOIN returns only those rows that have matching values in both tables. If there is no match, the row is not shown.

Example :-

```
SELECT e.employee_id, e.first_name, e.last_name, d.department_id, d.department_name
FROM employees3 e
INNER JOIN
departments  d
ON e.department_id=d.department_id;
```

Output :-

	employee_id integer	first_name character varying (50)	last_name character varying (50)	department_id integer	department_name character varying (50)
1	1	Rahul	Sharma	101	Sales
2	2	Priya	Mehta	102	Marketing
3	3	Ankit	Verma	103	IT
4	5	Aman	Singh	101	Sales

2. Left JOIN :-

- The LEFT JOIN returns all rows from the left table, and the matched rows from the right table.
- If there is no match in the right table, NULL values are returned for right-side columns.

Example :-

```
SELECT e.employee_id, e.first_name, e.last_name, d.department_id, d.department_name
FROM employees3 e
LEFT JOIN
departments  d
ON e.department_id=d.department_id;
```

Output :-

	employee_id integer	first_name character varying (50)	last_name character varying (50)	department_id integer	department_name character varying (50)
1	1	Rahul	Sharma	101	Sales
2	2	Priya	Mehta	102	Marketing
3	3	Ankit	Verma	103	IT
4	4	Simran	Kaur	[null]	[null]
5	5	Aman	Singh	101	Sales

3. RIGHT JOIN :-

- The RIGHT JOIN returns all rows from the right table, and the matched rows from the left table.
- If there is no match in the left table, NULL values are returned for left-side columns.

Example :- SELECT e.employee_id, e.first_name, e.last_name, d.department_id, d.department_name

```
FROM employees3 e    L
EFT JOIN
departments  d
ON e.department_id=d.department_id;
```

	employee_id integer	first_name character varying (50)	last_name character varying (50)	department_id integer	department_name character varying (50)
1	1	Rahul	Sharma	101	Sales
2	2	Priya	Mehta	102	Marketing
3	3	Ankit	Verma	103	IT
4	5	Aman	Singh	101	Sales
5	[null]	[null]	[null]	104	HR

4. FULL JOIN (FULL OUTER JOIN) :-

- The FULL JOIN returns all rows from both tables. If a row has no match in one of the tables, NULL is used for missing values.

Example :-

```
SELECT e.employee_id, e.first_name, e.last_name, d.department_id, d.department_name
FROM employees3 e
FULL OUTER JOIN
departments  d
ON e.department_id=d.department_id;
```

Output :-

	employee_id integer	first_name character varying (50)	last_name character varying (50)	department_id integer	department_name character varying (50)
1	1	Rahul	Sharma	101	Sales
2	2	Priya	Mehta	102	Marketing
3	3	Ankit	Verma	103	IT
4	4	Simran	Kaur	[null]	[null]
5	5	Aman	Singh	101	Sales
6	[null]	[null]	[null]	104	HR

5. CROSS JOIN :-

- The CROSS JOIN returns the cartesian product of two tables. That means every row of the first table is combined with every row of the second table.

Example :-

```
SELECT e.first_name, e.last_name, d.department_name
FROM employees3 e
CROSS JOIN
   departments d;
```

Output :-

	first_name character varying (50)	last_name character varying (50)	department_name character varying (50)
1	Rahul	Sharma	Sales
2	Priya	Mehta	Sales
3	Ankit	Verma	Sales
4	Simran	Kaur	Sales
5	Aman	Singh	Sales
6	Rahul	Sharma	Marketing
7	Priya	Mehta	Marketing
8	Ankit	Verma	Marketing
9	Simran	Kaur	Marketing
10	Aman	Singh	Marketing
11	Rahul	Sharma	IT
12	Priya	Mehta	IT
13	Ankit	Verma	IT
14	Simran	Kaur	IT
15	Aman	Singh	IT
16	Rahul	Sharma	HR
17	Priya	Mehta	HR
18	Ankit	Verma	HR
19	Simran	Kaur	HR
20	Aman	Singh	HR

7. SELF JOIN :-

- A Self Join is a type of JOIN where a table is joined with itself. It is used when rows within the same table are related to each other.

Example :-

```
SELECT e1.first_name AS Employee_name1,
    e2.first_name AS Employee_name2
FROM employees3 e1 JOIN employees3 e2
ON e1.department_id=e2.department_id AND e1.employee_id!=e2.employee_id
```

Output :-

	employee_name1 character varying (50)	employee_name2 character varying (50)
1	Rahul	Aman
2	Aman	Rahul

Example :-

```
SELECT e1.first_name AS Employee_name1,
    e2.first_name AS Employee_name2,
    d.department_name
FROM employees3 e1 JOIN employees3 e2
ON e1.department_id=e2.department_id AND e1.employee_id!=e2.employee_id
JOIN
departments d
ON   e1.department_id=d.department_id;
```

	employee_name1 character varying (50)	employee_name2 character varying (50)	department_name character varying (50)
1	Rahul	Aman	Sales
2	Aman	Rahul	Sales

Project :-

```
-- Create Database
CREATE DATABASE OnlineBookstore;
-- Create Tables
DROP TABLE IF EXISTS Books;
CREATE TABLE Books (
    Book_ID SERIAL PRIMARY KEY,
    Title VARCHAR(100),
    Author VARCHAR(100),
    Genre VARCHAR(50),
    Published_Year INT,
    Price NUMERIC(10, 2),
    Stock INT
);
```

```
DROP TABLE IF EXISTS customers;
CREATE TABLE Customers (
    Customer_ID SERIAL PRIMARY KEY,
    Name VARCHAR(100),
    Email VARCHAR(100),
    Phone VARCHAR(15),
    City VARCHAR(50),
    Country VARCHAR(150)
);

DROP TABLE IF EXISTS orders;
CREATE TABLE Orders (
    Order_ID SERIAL PRIMARY KEY,
    Customer_ID INT REFERENCES Customers(Customer_ID),
    Book_ID INT REFERENCES Books(Book_ID),
    Order_Date DATE,
    Quantity INT,
    Total_Amount NUMERIC(10, 2)
);

SELECT * FROM Books;
SELECT * FROM Customers;
SELECT * FROM Orders;

-- Import Data into Books Table
COPY Books(Book_ID, Title, Author, Genre, Published_Year, Price, Stock)
FROM 'D:\Course Updates\30 Day Series\SQL\CSV\Books.csv'
CSV HEADER;

-- Import Data into Customers Table
COPY Customers(Customer_ID, Name, Email, Phone, City, Country)
FROM 'D:\Course Updates\30 Day Series\SQL\CSV\Customers.csv'
CSV HEADER;

-- Import Data into Orders Table
COPY Orders(Order_ID, Customer_ID, Book_ID, Order_Date, Quantity, Total_Amount)
FROM 'D:\Course Updates\30 Day Series\SQL\CSV\Orders.csv'
CSV HEADER;
```

Basic Queries :-

-- 1) Retrieve all books in the "Fiction" genre :-

```
SELECT * FROM Books WHERE Genre = 'Fiction' ;
```

-- 2) Find books published after the year 1950 :-

```
SELECT * FROM Books WHERE Published_Year > 1950 ;
```

-- 3) List all customers from the Canada :-

```
SELECT * FROM Customers WHERE Country = 'Canada' ;
```

-- 4) Show orders placed in November 2023 :-

```
SELECT * FROM Orders WHERE order_date BETWEEN '2023-11-01' AND '2023-11-30' ;
```

-- 5) Retrieve the total stock of books available :-

```
SELECT SUM( stock ) AS Total_stock From Books ;
```

-- 6) Find the details of the most expensive book :-

```
SELECT * FROM Books  ORDER BY Price DESC  LIMIT 1 ;
```

-- 7) Show all customers who ordered more than 1 quantity of a book :-

```
SELECT * FROM Orders WHERE quantity > 1 ;
```

-- 8) Retrieve all orders where the total amount exceeds $20 :-

```
SELECT * FROM Orders WHERE total_amount > 20 ;
```

-- 9) List all genres available in the Books table :-

```
SELECT DISTINCT genre From Books;
```

-- 10) Find the book with the lowest stock :-

```
SELECT * FROM Books ORDER BY Stock ASC LIMIT 1 ;
```

-- 11) Calculate the total revenue generated from all orders :-

```
SELECT SUM(total_amount) AS Revenue FROM Orders ;
```

Advance Queries :-

-- 1) Retrieve the total number of books sold for each genre :-

```
SELECT b.Genre , sum( o.Quantity ) As Total_Book_Sold
FROM Orders  o
JOIN  BOOKS  b   ON   o.book_id = b.book_id
GROUP BY b.Genre;
```

-- 2) Find the average price of books in the "Fantasy" genre :-

```
SELECT AVG ( price ) As Average_price
FROM Books
Where Genre = 'Fantasy' ;
```

-- 3) List customers who have placed at least 2 orders :-

```
SELECT  Customer_id , COUNT (Order_id) AS ORDER_COUNT  From orders
GROUP BY customer_id
HAVING COUNT (Order_id) >= 2;
```

Add Customer :-

```
SELECT  o.customer_id , c.name, COUNT (o.Order_id) AS ORDER_COUNT
 From orders o   JOIN   customer  c   ON    o.customer_id = c.customer_id
GROUP BY o.customer_id , c.name
HAVING COUNT (Order_id) >= 2;
```

-- 4) Find the most frequently ordered book :-

```
SELECT Book_id, COUNT ( order_id ) AS ORDER_COUNT From orders
GROUP BY Book_id
ORDER BY ORDER_COUNT DESC LIMIT 1;
```

-- 5) Show the top 3 most expensive books of 'Fantasy' Genre :-

```
SELECT * FROM books
WHERE genre = 'Fantasy'
ORDE BY Price  DESC  LIMIT 3;
```

-- 6) Retrieve the total quantity of books sold by each author :-

```
SELECT b.author, SUM (o.quantity) AS Total_Book_sold
FROM orders o
JOIN books  b   ON   o.book_id = b.book_id
GROUP BY  b.Author;
```

-- 7) List the cities where customers who spent over $30 are located :-

```
SELECT DISTINCT c.city  , total_amount                    //Distinct Represent Unique value
FROM orders  o
JOIN customers  c  ON  o.customer_id = c.customer_id
WHERE o.total_amount > 30;
```

-- 8) Find the customer who spent the most on orders :-

```
SELECT  c.customer_id , c.name , SUM ( o.total_amount ) AS Total_spent
From orders  o
JOIN  customers c  ON  o.customer_id = c.customer_id
GROUP BY c.csutomer_id , c.name
ORDER BY  Total_spent Desc LIMIT 1;
```

--9) Calculate the stock remaining after fulfilling all orders :-

```
SELECT  b.book_id , b.title , b.stock , COALESCE(SUM(o.quantity), 0 ) AS Order_quantity, b.stock - COALESCE(SUM(o.quantity), 0 ) AS Remaining_quantity
From books b
LEFT JOIN orders  o  ON  b.book_id = o.book_id
GROUP BY b.book_id;
```

COALESCE In Sql :-

- A COALESCE is a SQL function that returns the first non-NULL value from a list of values. It is often used to handle NULL values in queries.

Example :-

```
CREATE TABLE employees (  id INT,  first_name VARCHAR(50),  last_name VARCHAR(50) );
INSERT INTO employees (id, first_name, last_name) VALUES  (1, 'John', NULL),  (2, NULL, 'Patel'),  (3, NULL, NULL);
SELECT   id,  COALESCE(first_name, last_name, 'No Name') AS name  FROM employees;
```

Book Name :-

```
SELECT o.Book_id, b.title, COUNT ( o.order_id ) AS ORDER_COUNT
From orders o   JOINS  books  b     ON   o.book_id = b.book_id
GROUP BY o.book_id , b.title
ORDER BY ORDER_COUNT  DESC  LIMIT 1;
```