DATABASE - POSTGRESQL

- UNDERSTANDING BASICS

1. What is a Database ?

An Organized collection of data. A method to manipulate and access the data.

2. Database VS DBMS ?

An App sends the information and stores it in a database. DBMS and Database works hand in hand. DBMS is used to store and retrieve the data from the database in organized way. Example of DBMS are PostgreSql, MySQL, Oracle etc.

3. What is RDBMS ?

A type of database system that stores data in structured tables (using rows and columns) and uses SQL for managing and querying data.

4. SQL vs PostgreSQL

SQL → Structured Query Language Which is used to talk to our Database.

Ex - SELECT * FROM person_db;

PostgreSql → The Main work of PostgreSql is take the data from application and store it in database for that it used Structured Query language(SQL)

- INSTALLATION WINDOWS

1. Go to the following link - Download click on Windows and then download the installer.
2. Select the latest version (17.0) for Windows and click download. Follow the installation process.
3. Seach Pgadmin in taskbar and open it. It is like a UI for Posgres where we can write our queries.
4. We can use SQL shell to type Queries.
5. \list → list all the databases
6. \! cls → clear all the queries.
7. Create Database → CREATE DATABASE TEST;
8. How to access through CMD

```
Auto (SQL)
Copy the path `C:\Program Files\PostgreSQL\17\bin` and set it in env variables
Open the CMD and type
psql -U postgres
```

```
password: root (and you are connected)

how to connect to the database "test" as user postgres
-> \connect test;
```

- ## Database Vs Schema Vs Table

Lets take an example of Instagram Database. Database is a high level overview like in case of Instagram we can call its Database as Instagram_DB. Then we know instagram have many features like user, profiles, followers, post, reels etc. We can group few things and call it as a Schema. Then each schema wil have its seperate Table to store data.

- ## Database And CRUD

1. Working with Databases

1.1 List Down existing database

→ `\list` - (On psql shell) - command line

→ In Pgadmin using query

```SQL
SELECT datname FROM pg_database;
```

1.2 Creating a new Database

```SQL
CREATE DATABASE <db_name>;

ex - CREATE DATABASE test;
```

→ Right click on Database in Pgadmin and click create. Give a name to the database and click save.

1.3 Change a Database

→ `\c instadb` (psql shell) - command line

`You are now connected to database "instadb" as user "postgres".`

→ In Pgadmin click on any Database for which we want to run the query and then click on `Query tool`

1.4 Deleting a Database

```sql
DROP DATABASE test;
```

- CRUD operation (Create, Read, Update, Delete)

1.1 Creating Tables

→ A table is a collection of Related data held in a table format within a database.

Create a Database called `persondb` and open query tool and type this to create a table in persondb database. This will create a table person with three columns (id, name, city)

```sql
CREATE TABLE person(
        id INT,
        name VARCHAR(100),
        city VARCHAR(100)
);
```

To verify (using pgadmin)

```sql
Click on
persondb -> Schemas -> public -> Tables -> person -> Columns
```

To verify (using psql shell)

```
\c persondb;
\d person;

(output)

 Column |          Type          | Collation | Nullable | Default
--------+------------------------+-----------+----------+---------
 id     | integer                |           |          |
 name   | character varying(100) |           |          |
 city   | character varying(100) |           |          |
```

1.2 Inserting Data In Table

```sql
INSERT INTO person(id, name, city)
VALUES(101, 'Raju', 'Delhi');

/* multiple data */

INSERT INTO person(id, name, city)
VALUES
(102, 'Sham', 'Mumbai'),
(103, 'Paul', 'Chennai');

/* no need to specify column name as long as we are adding the data for all
columns in sequence */

INSERT INTO person
VALUES(104, 'Alex', 'Pune');
```

1.3 Reading Data from Table

```sql
SELECT * FROM person; /* All Column */
```

Data Output | Messages | Notifications

	id integer	name character varying (100)	city character varying (100)
1	101	Raju	Delhi
2	102	Sham	Mumbai
3	103	Paul	Chennai
4	104	Alex	Pune

→ If we want to see only one column or multiple column

```sql
SELECT name FROM person;      /* one column */
```

```
SELECT name, city FROM person; /* multiple column */
```

Data Output Messages Notifications

	name character varying (100)
1	Raju
2	Sham
3	Paul
4	Alex

Data Output Messages Notifications

	name character varying (100)	city character varying (100)
1	Raju	Delhi
2	Sham	Mumbai
3	Paul	Chennai
4	Alex	Pune

1.4 Updating Data From Table

SQL

```
UPDATE person
SET city = 'Banglore'
WHERE id = 101;

SELECT * FROM person;
```

Data Output Messages Notifications

	id integer	name character varying (100)	city character varying (100)
1	102	Sham	Mumbai
2	103	Paul	Chennai
3	104	Alex	Pune
4	101	Raju	Banglore

1.5 Deleting Data From Table

SQL

```
DELETE FROM person
WHERE id = 104;

SELECT * FROM person;
```

Data Output | Messages | Notifications

SQL

	id integer	name character varying (100)	city character varying (100)
1	102	Sham	Mumbai
2	103	Paul	Chennai
3	101	Raju	Banglore

- DrawBack of Current Structure

```sql
INSERT INTO person
VALUES(101, 'Alex', 'Banglore');

SELECT * FROM person;
```

Data Output | Messages | Notifications

SQL

	id integer	name character varying (100)	city character varying (100)
1	102	Sham	Mumbai
2	103	Paul	Chennai
3	101	Raju	Banglore
4	101	Alex	Banglore

Raju and Alex both are having same id now. So even if we add duplicate id it is still accepting it.

```sql
INSERT INTO person(id, name)
VALUES(105, 'Victa');

SELECT * FROM person;
```

Data Output | Messages | Notifications

SQL

	id integer	name character varying (100)	city character varying (100)
1	102	Sham	Mumbai
2	103	Paul	Chennai
3	101	Raju	Banglore
4	101	Alex	Banglore
5	105	Victa	[null]

When we don't provide any value for city for a user, NULL value is set. We have to deal with that as well.

- DataTypes

→ An Attribute that specifies the type of data in a column of our database - table

Most Widely used are -

1. Numeric → INT DOUBLE FLOAT DECIMAL
2. String → VARCHAR
3. Date → DATE
4. Boolean → BOOLEAN

Auto (Haskell)

```
DECIMAL(5,2)
5-> Total digit
2-> Digits after decimal

example - 155.38, 119.12, 28.15, 1150.15(x)
```

- Constraint

A constraint in PostgreSQL is a rule applied to a Column.

1.1 Primary Key

1. The PRIMARY KEY constraint uniquely identifies each record in a table. (eg - in case of instagram_db, username can be set as primary key)
2. Primary keys must contain UNIQUE values, and cannot contain NULL values.
3. A table can have only one primary key

1.2 NOT NULL constraint

→ For a particular column, null value is not allowed.

```sql
CREATE TABLE customers(
    id INT NOT NULL,
    name VARCHAR(100) NOT NULL
);
```

1.3 DEFAULT value

→ When No value is provided for a particular column, default value will be set.

```sql
CREATE TABLE customers(
    acc_no INT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    acc_type VARCHAR(50) NOT NULL DEFAULT 'Savings'
);
```

1.4 AUTO_INCREMENT (SERIAL)

→ No need to set the value for a particular column where we are auto incrementaing the value. For every record value will be incremented and it will be set (Eg - 1,2,3,4...)

```sql
CREATE TABLE employees (
    id SERIAL PRIMARY KEY,
    firstname VARCHAR(50),
    lastname VARCHAR(50)
);
```

- TASK 1: Create New Table

1. Create a Database called bank_db and create a table called employees

```sql
CREATE TABLE employees (
```

```
        emp_id SERIAL PRIMARY KEY,
        fname VARCHAR(100) NOT NULL,
        lname VARCHAR(100) NOT NULL,
        email VARCHAR(100) NOT NULL UNIQUE,
        dept VARCHAR(50),
        salary DECIMAL(10,2) DEFAULT 30000.00,
        hire_date DATE NOT NULL DEFAULT CURRENT_DATE
);
```

2. Add Data into it

SQL

```
INSERT INTO employees (emp_id, fname, lname, email, dept, salary, hire_date)
VALUES(1, 'Raj', 'Sharma', 'raj.sharma@example.com', 'IT', 50000.00, '2020-01-
15');
```

SQL

```
INSERT INTO employees (fname, lname, email, dept)
VALUES('Priya', 'Singh', 'priya.singh@example.com', 'HR',);
```

error -

Bash

```
ERROR:  Key (emp_id)=(1) already exists.duplicate key value violates unique
constraint "employees_pkey"

ERROR:  duplicate key value violates unique constraint "employees_pkey"
SQL state: 23505
Detail: Key (emp_id)=(1) already exists.
```

solution

Auto (SQL)

```
 SELECT setval('employees_emp_id_seq', 1);

SELECT currval('employees_emp_id_seq');
 currval
---------
       1
(1 row)
```

run the query again

SQL

```sql
INSERT INTO employees (fname, lname, email, dept)
VALUES('Priya', 'Singh', 'priya.singh@example.com', 'HR',);
```

```
bank_db=# SELECT * FROM employees;
 emp_id | fname | lname  |          email          | dept |  salary  | hire_date
--------+-------+--------+-------------------------+------+----------+------------
      1 | Raj   | Sharma | raj.sharma@example.com  | IT   | 50000.00 | 2020-01-15
      2 | Priya | Singh  | priya.singh@example.com | HR   | 30000.00 | 2024-11-05
(2 rows)
```

Add all the data

Auto (SQL)

```sql
INSERT INTO employees (emp_id, fname, lname, email, dept, salary, hire_date)
VALUES
(1, 'Raj', 'Sharma', 'raj.sharma@example.com', 'IT', 50000.00, '2020-01-15'),
(2, 'Priya', 'Singh', 'priya.singh@example.com', 'HR', 45000.00, '2019-03-22'),
(3, 'Arjun', 'Verma', 'arjun.verma@example.com', 'IT', 55000.00, '2021-06-01'),
(4, 'Suman', 'Patel', 'suman.patel@example.com', 'Finance', 60000.00, '2018-07-30'),
(5, 'Kavita', 'Rao', 'kavita.rao@example.com', 'HR', 47000.00, '2020-11-10'),
(6, 'Amit', 'Gupta', 'amit.gupta@example.com', 'Marketing', 52000.00, '2020-09-25'),
(7, 'Neha', 'Desai', 'neha.desai@example.com', 'IT', 48000.00, '2019-05-18'),
(8, 'Rahul', 'Kumar', 'rahul.kumar@example.com', 'IT', 53000.00, '2021-02-14'),
(9, 'Anjali', 'Mehta', 'anjali.mehta@example.com', 'Finance', 61000.00, '2018-12-03'),
(10, 'Vijay', 'Nair', 'vijay.nair@example.com', 'Marketing', 50000.00, '2020-04-19');
```

```
bank_db=# SELECT * FROM employees;
 emp_id | fname  | lname  |          email          |   dept    |  salary  | hire_date
--------+--------+--------+-------------------------+-----------+----------+------------
      1 | Raj    | Sharma | raj.sharma@example.com  | IT        | 50000.00 | 2020-01-15
      2 | Priya  | Singh  | priya.singh@example.com | HR        | 45000.00 | 2019-03-22
      3 | Arjun  | Verma  | arjun.verma@example.com | IT        | 55000.00 | 2021-06-01
      4 | Suman  | Patel  | suman.patel@example.com | Finance   | 60000.00 | 2018-07-30
      5 | Kavita | Rao    | kavita.rao@example.com  | HR        | 47000.00 | 2020-11-10
      6 | Amit   | Gupta  | amit.gupta@example.com  | Marketing | 52000.00 | 2020-09-25
      7 | Neha   | Desai  | neha.desai@example.com  | IT        | 48000.00 | 2019-05-18
      8 | Rahul  | Kumar  | rahul.kumar@example.com | IT        | 53000.00 | 2021-02-14
      9 | Anjali | Mehta  | anjali.mehta@example.com| Finance   | 61000.00 | 2018-12-03
     10 | Vijay  | Nair   | vijay.nair@example.com  | Marketing | 50000.00 | 2020-04-19
(10 rows)
```

- Data Refining

Clauses

1. Where
2. Distinct
3. Order By
4. Limit
5. Like

1.1 WHERE

SQL

```sql
SELECT * FROM employees
WHERE emp_id = 5;
```

```
 emp_id | fname  | lname |         email          | dept |  salary  | hire_date
--------+--------+-------+------------------------+------+----------+------------
      5 | Kavita | Rao   | kavita.rao@example.com | HR   | 47000.00 | 2020-11-10
(1 row)
```

SQL

```sql
SELECT * FROM employees
WHERE dept = 'HR';
```

```
 emp_id | fname  | lname |          email          | dept |  salary  | hire_date
--------+--------+-------+-------------------------+------+----------+------------
      2 | Priya  | Singh | priya.singh@example.com | HR   | 45000.00 | 2019-03-22
      5 | Kavita | Rao   | kavita.rao@example.com  | HR   | 47000.00 | 2020-11-10
(2 rows)
```

SQL

```sql
SELECT * FROM employees
WHERE salary >= 50000;
```

```
 emp_id | fname  | lname  |          email          |   dept    |  salary  | hire_date
--------+--------+--------+-------------------------+-----------+----------+------------
      1 | Raj    | Sharma | raj.sharma@example.com  | IT        | 50000.00 | 2020-01-15
      3 | Arjun  | Verma  | arjun.verma@example.com | IT        | 55000.00 | 2021-06-01
      4 | Suman  | Patel  | suman.patel@example.com | Finance   | 60000.00 | 2018-07-30
      6 | Amit   | Gupta  | amit.gupta@example.com  | Marketing | 52000.00 | 2020-09-25
      8 | Rahul  | Kumar  | rahul.kumar@example.com | IT        | 53000.00 | 2021-02-14
      9 | Anjali | Mehta  | anjali.mehta@example.com| Finance   | 61000.00 | 2018-12-03
     10 | Vijay  | Nair   | vijay.nair@example.com  | Marketing | 50000.00 | 2020-04-19
(7 rows)
```

SQL

```sql
SELECT * FROM employees
WHERE dept = 'HR' or dept = 'Finance';
```

```
 emp_id | fname  | lname |          email          |  dept   |  salary  | hire_date
--------+--------+-------+-------------------------+---------+----------+------------
      2 | Priya  | Singh | priya.singh@example.com | HR      | 45000.00 | 2019-03-22
      4 | Suman  | Patel | suman.patel@example.com | Finance | 60000.00 | 2018-07-30
      5 | Kavita | Rao   | kavita.rao@example.com  | HR      | 47000.00 | 2020-11-10
      9 | Anjali | Mehta | anjali.mehta@example.com| Finance | 61000.00 | 2018-12-03
(4 rows)
```

SQL

```sql
SELECT * FROM employees
WHERE dept = 'IT' AND salary > 50000;
```

```
 emp_id | fname | lname |          email          | dept |  salary  | hire_date
--------+-------+-------+-------------------------+------+----------+------------
      3 | Arjun | Verma | arjun.verma@example.com | IT   | 55000.00 | 2021-06-01
      8 | Rahul | Kumar | rahul.kumar@example.com | IT   | 53000.00 | 2021-02-14
(2 rows)
```

SQL

```sql
SELECT * FROM employees
WHERE dept = 'IT' or dept = 'HR' or dept = 'Finance';

Best way to do is ->

SELECT * FROM employees
WHERE dept IN('IT', 'HR', 'Finance');
```

```
bank_db-# WHERE dept IN('IT', 'HR', 'Finance');
 emp_id | fname  | lname  |          email          |  dept   |  salary  | hire_date
--------+--------+--------+-------------------------+---------+----------+------------
      1 | Raj    | Sharma | raj.sharma@example.com  | IT      | 50000.00 | 2020-01-15
      2 | Priya  | Singh  | priya.singh@example.com | HR      | 45000.00 | 2019-03-22
      3 | Arjun  | Verma  | arjun.verma@example.com | IT      | 55000.00 | 2021-06-01
      4 | Suman  | Patel  | suman.patel@example.com | Finance | 60000.00 | 2018-07-30
      5 | Kavita | Rao    | kavita.rao@example.com  | HR      | 47000.00 | 2020-11-10
      7 | Neha   | Desai  | neha.desai@example.com  | IT      | 48000.00 | 2019-05-18
      8 | Rahul  | Kumar  | rahul.kumar@example.com | IT      | 53000.00 | 2021-02-14
      9 | Anjali | Mehta  | anjali.mehta@example.com| Finance | 61000.00 | 2018-12-03
(8 rows)
```

SQL

```sql
SELECT * FROM employees
WHERE dept NOT IN('IT', 'HR', 'Finance');
```

```
 emp_id | fname | lname |          email          |    dept    |  salary  | hire_date
--------+-------+-------+-------------------------+------------+----------+------------
      6 | Amit  | Gupta | amit.gupta@example.com  | Marketing  | 52000.00 | 2020-09-25
     10 | Vijay | Nair  | vijay.nair@example.com  | Marketing  | 50000.00 | 2020-04-19
(2 rows)
```

SQL

```
SELECT * FROM employees
WHERE salary BETWEEN 50000 AND 60000;
```

```
 emp_id | fname | lname  |          email          |   dept    |  salary  | hire_date
--------+-------+--------+-------------------------+-----------+----------+------------
      1 | Raj   | Sharma | raj.sharma@example.com  | IT        | 50000.00 | 2020-01-15
      3 | Arjun | Verma  | arjun.verma@example.com | IT        | 55000.00 | 2021-06-01
      4 | Suman | Patel  | suman.patel@example.com | Finance   | 60000.00 | 2018-07-30
      6 | Amit  | Gupta  | amit.gupta@example.com  | Marketing | 52000.00 | 2020-09-25
      8 | Rahul | Kumar  | rahul.kumar@example.com | IT        | 53000.00 | 2021-02-14
     10 | Vijay | Nair   | vijay.nair@example.com  | Marketing | 50000.00 | 2020-04-19
(6 rows)
```

1.2 DISTINCT

SQL

```
SELECT dept FROM employees;
```

```
   dept
-----------
 IT
 HR
 IT
 Finance
 HR
 Marketing
 IT
 IT
 Finance
 Marketing
(10 rows)
```

But i want distinct no of departments, currently there are lot of duplicate values.

SQL

```
SELECT DISTINCT dept FROM employees;
```

```
   dept
-----------
 Marketing
 Finance
 IT
 HR
(4 rows)
```

1.3 ORDER BY

→ ORDER BY is used for sorting of data

SQL

```sql
SELECT * FROM employees ORDER BY fname;
```

```
 emp_id | fname  | lname  |          email          |   dept    |  salary  | hire_date
--------+--------+--------+-------------------------+-----------+----------+------------
      6 | Amit   | Gupta  | amit.gupta@example.com  | Marketing | 52000.00 | 2020-09-25
      9 | Anjali | Mehta  | anjali.mehta@example.com| Finance   | 61000.00 | 2018-12-03
      3 | Arjun  | Verma  | arjun.verma@example.com | IT        | 55000.00 | 2021-06-01
      5 | Kavita | Rao    | kavita.rao@example.com  | HR        | 47000.00 | 2020-11-10
      7 | Neha   | Desai  | neha.desai@example.com  | IT        | 48000.00 | 2019-05-18
      2 | Priya  | Singh  | priya.singh@example.com | HR        | 45000.00 | 2019-03-22
      8 | Rahul  | Kumar  | rahul.kumar@example.com | IT        | 53000.00 | 2021-02-14
      1 | Raj    | Sharma | raj.sharma@example.com  | IT        | 50000.00 | 2020-01-15
      4 | Suman  | Patel  | suman.patel@example.com | Finance   | 60000.00 | 2018-07-30
     10 | Vijay  | Nair   | vijay.nair@example.com  | Marketing | 50000.00 | 2020-04-19
(10 rows)
```

SQL

```sql
SELECT * FROM employees ORDER BY fname DESC;
```

```
 emp_id | fname  | lname  |          email           |   dept    |  salary  | hire_date
--------+--------+--------+--------------------------+-----------+----------+------------
     10 | Vijay  | Nair   | vijay.nair@example.com   | Marketing | 50000.00 | 2020-04-19
      4 | Suman  | Patel  | suman.patel@example.com  | Finance   | 60000.00 | 2018-07-30
      1 | Raj    | Sharma | raj.sharma@example.com   | IT        | 50000.00 | 2020-01-15
      8 | Rahul  | Kumar  | rahul.kumar@example.com  | IT        | 53000.00 | 2021-02-14
      2 | Priya  | Singh  | priya.singh@example.com  | HR        | 45000.00 | 2019-03-22
      7 | Neha   | Desai  | neha.desai@example.com   | IT        | 48000.00 | 2019-05-18
      5 | Kavita | Rao    | kavita.rao@example.com   | HR        | 47000.00 | 2020-11-10
      3 | Arjun  | Verma  | arjun.verma@example.com  | IT        | 55000.00 | 2021-06-01
      9 | Anjali | Mehta  | anjali.mehta@example.com | Finance   | 61000.00 | 2018-12-03
      6 | Amit   | Gupta  | amit.gupta@example.com   | Marketing | 52000.00 | 2020-09-25
(10 rows)
```

SQL

```sql
SELECT * FROM employees ORDER BY emp_id DESC;
```

```
 emp_id | fname  | lname  |          email           |   dept    |  salary  | hire_date
--------+--------+--------+--------------------------+-----------+----------+------------
     10 | Vijay  | Nair   | vijay.nair@example.com   | Marketing | 50000.00 | 2020-04-19
      9 | Anjali | Mehta  | anjali.mehta@example.com | Finance   | 61000.00 | 2018-12-03
      8 | Rahul  | Kumar  | rahul.kumar@example.com  | IT        | 53000.00 | 2021-02-14
      7 | Neha   | Desai  | neha.desai@example.com   | IT        | 48000.00 | 2019-05-18
      6 | Amit   | Gupta  | amit.gupta@example.com   | Marketing | 52000.00 | 2020-09-25
      5 | Kavita | Rao    | kavita.rao@example.com   | HR        | 47000.00 | 2020-11-10
      4 | Suman  | Patel  | suman.patel@example.com  | Finance   | 60000.00 | 2018-07-30
      3 | Arjun  | Verma  | arjun.verma@example.com  | IT        | 55000.00 | 2021-06-01
      2 | Priya  | Singh  | priya.singh@example.com  | HR        | 45000.00 | 2019-03-22
      1 | Raj    | Sharma | raj.sharma@example.com   | IT        | 50000.00 | 2020-01-15
(10 rows)
```

1.4 LIMIT

→ Show me only the first 3 rows

SQL

```sql
SELECT * FROM employees LIMIT 3;
```

1.5 LIKE

Give me the employee details whose first name starts with A

SQL

```
SELECT * FROM employees WHERE fname LIKE 'A%';
```

```
 emp_id | fname  | lname |          email           |   dept    |  salary  | hire_date
--------+--------+-------+--------------------------+-----------+----------+------------
      3 | Arjun  | Verma | arjun.verma@example.com  | IT        | 55000.00 | 2021-06-01
      6 | Amit   | Gupta | amit.gupta@example.com   | Marketing | 52000.00 | 2020-09-25
      9 | Anjali | Mehta | anjali.mehta@example.com | Finance   | 61000.00 | 2018-12-03
(3 rows)
```

Give me the employee details whose first name ends with a

SQL

```
SELECT * FROM employees WHERE fname LIKE '%a';
```

```
 emp_id | fname  | lname |          email           | dept |  salary  | hire_date
--------+--------+-------+--------------------------+------+----------+------------
      2 | Priya  | Singh | priya.singh@example.com  | HR   | 45000.00 | 2019-03-22
      5 | Kavita | Rao   | kavita.rao@example.com   | HR   | 47000.00 | 2020-11-10
      7 | Neha   | Desai | neha.desai@example.com   | IT   | 48000.00 | 2019-05-18
(3 rows)
```

Give me the employee details whose first name contains 'i' in between

SQL

```
SELECT * FROM employees WHERE fname LIKE '%i%';
```

```
 emp_id | fname  | lname |          email           |   dept    |  salary  | hire_date
--------+--------+-------+--------------------------+-----------+----------+------------
      2 | Priya  | Singh | priya.singh@example.com  | HR        | 45000.00 | 2019-03-22
      5 | Kavita | Rao   | kavita.rao@example.com   | HR        | 47000.00 | 2020-11-10
      6 | Amit   | Gupta | amit.gupta@example.com   | Marketing | 52000.00 | 2020-09-25
      9 | Anjali | Mehta | anjali.mehta@example.com | Finance   | 61000.00 | 2018-12-03
     10 | Vijay  | Nair  | vijay.nair@example.com   | Marketing | 50000.00 | 2020-04-19
(5 rows)
```

Give me the employee details whose dept has only two characters

SQL

```
SELECT * FROM employees WHERE dept LIKE '__';
```

```
 emp_id | fname  | lname  |          email          | dept |  salary  | hire_date
--------+--------+--------+-------------------------+------+----------+------------
      1 | Raj    | Sharma | raj.sharma@example.com  | IT   | 50000.00 | 2020-01-15
      2 | Priya  | Singh  | priya.singh@example.com | HR   | 45000.00 | 2019-03-22
      3 | Arjun  | Verma  | arjun.verma@example.com | IT   | 55000.00 | 2021-06-01
      5 | Kavita | Rao    | kavita.rao@example.com  | HR   | 47000.00 | 2020-11-10
      7 | Neha   | Desai  | neha.desai@example.com  | IT   | 48000.00 | 2019-05-18
      8 | Rahul  | Kumar  | rahul.kumar@example.com | IT   | 53000.00 | 2021-02-14
(6 rows)
```

Give me the employee details whose first name has a as a second character

```sql
SELECT * FROM employees WHERE fname LIKE '_a%';
```

```
 emp_id | fname  | lname  |          email          | dept |  salary  | hire_date
--------+--------+--------+-------------------------+------+----------+------------
      1 | Raj    | Sharma | raj.sharma@example.com  | IT   | 50000.00 | 2020-01-15
      5 | Kavita | Rao    | kavita.rao@example.com  | HR   | 47000.00 | 2020-11-10
      8 | Rahul  | Kumar  | rahul.kumar@example.com | IT   | 53000.00 | 2021-02-14
(3 rows)
```

- Aggregate Function

How to find total no of employees ?

Employee with max salary

Avg salary of employees

1. COUNT
2. SUM
3. AVG
4. MIN
5. MAX

1.1 COUNT

Find the total no of employees.

```sql
SELECT COUNT(emp_id) FROM employees;
```

```
 count
-------
    10
(1 row)
```

1.2 SUM

How much total salary bank is spending on employees in total

```sql
SELECT SUM(salary) FROM employees;
```

```
    sum
-----------
 521000.00
(1 row)
```

1.3 AVG

Avg salary of employees

```sql
SELECT AVG(salary) FROM employees;
```

```
        avg
--------------------
 52100.000000000000
(1 row)
```

1.4 MIN

```sql
SELECT MIN(salary) FROM employees;
```

1.5 MAX

```sql
SELECT MAX(salary) FROM employees;
```

- What is Group By

Find out the number of employees in each department.

```sql
SELECT dept, COUNT(emp_id) from employees GROUP BY dept;
```

```
   dept    | count
-----------+-------
 Marketing |     2
 Finance   |     2
 IT        |     4
 HR        |     2
(4 rows)
```

Find out the total salary spend per department on employees.

```sql
SELECT dept, SUM(salary) FROM employees GROUP BY dept;
```

```
   dept    |    sum
-----------+-----------
 Marketing | 102000.00
 Finance   | 121000.00
 IT        | 206000.00
 HR        |  92000.00
(4 rows)
```

Find out the max salary of employee per department

```sql
SELECT dept, MAX(salary) FROM employees GROUP BY dept;
```

```
   dept     |   max
------------+----------
 Marketing  | 52000.00
 Finance    | 61000.00
 IT         | 55000.00
 HR         | 47000.00
(4 rows)
```

- String Functions

1.1 CONCAT

SQL

```sql
SELECT CONCAT('Hello', 'World');
```

```
   concat
------------
 HelloWorld
(1 row)
```

SQL

```sql
SELECT CONCAT(fname,' ',lname) AS fullname FROM employees;
```

```
   fullname
--------------
 Raj Sharma
 Priya Singh
 Arjun Verma
 Suman Patel
 Kavita Rao
 Amit Gupta
 Neha Desai
 Rahul Kumar
 Anjali Mehta
 Vijay Nair
(10 rows)
```

```sql
SELECT emp_id, CONCAT(fname,' ',lname) AS fullname, dept FROM employees;
```

```
 emp_id |   fullname    |   dept
--------+---------------+-----------
      1 | Raj Sharma    | IT
      2 | Priya Singh   | HR
      3 | Arjun Verma   | IT
      4 | Suman Patel   | Finance
      5 | Kavita Rao    | HR
      6 | Amit Gupta    | Marketing
      7 | Neha Desai    | IT
      8 | Rahul Kumar   | IT
      9 | Anjali Mehta  | Finance
     10 | Vijay Nair    | Marketing
(10 rows)
```

1.2 CONCAT_WS (Concat with seperator)

```sql
SELECT CONCAT_WS('-',fname,lname) AS fullname FROM employees;
```

```
   fullname
--------------
 Raj-Sharma
 Priya-Singh
 Arjun-Verma
 Suman-Patel
 Kavita-Rao
 Amit-Gupta
 Neha-Desai
 Rahul-Kumar
 Anjali-Mehta
 Vijay-Nair
(10 rows)
```

```sql
```

```
SELECT emp_id, CONCAT_WS(' ', fname, lname) AS fullname, dept FROM employees;
```

```
 emp_id |   fullname    |   dept
--------+---------------+-----------
      1 | Raj Sharma    | IT
      2 | Priya Singh   | HR
      3 | Arjun Verma   | IT
      4 | Suman Patel   | Finance
      5 | Kavita Rao    | HR
      6 | Amit Gupta    | Marketing
      7 | Neha Desai    | IT
      8 | Rahul Kumar   | IT
      9 | Anjali Mehta  | Finance
     10 | Vijay Nair    | Marketing
(10 rows)
```

1.3 SUBSTRING

SQL

```
SELECT SUBSTR('Hello World', 1, 5);
```

```
 substr
--------
 Hello
(1 row)
```

1.4 REPLACE

SQL

```
SELECT REPLACE('abc xyz', 'abc', 'pqr');
```

```
 replace
---------
 pqr xyz
(1 row)
```

```sql
SELECT REPLACE(dept, 'IT', 'TECH') from employees;
```

```
 replace
-----------
 TECH
 HR
 TECH
 Finance
 HR
 Marketing
 TECH
 TECH
 Finance
 Marketing
(10 rows)
```

1.5 REVERSE

```sql
SELECT REVERSE('HELLO');
```

```
 reverse
---------
 OLLEH
(1 row)
```

```sql
SELECT REVERSE(fname) FROM employees;
```

```
 reverse
---------
 jaR
 ayirP
 nujrA
 namuS
 ativaK
 timA
 aheN
 luhaR
 ilajnA
 yajiV
(10 rows)
```

1.6 LENGTH

```sql
SELECT LENGTH('Hello');
```

```
 length
--------
      5
(1 row)
```

```sql
SELECT LENGTH(fname) FROM employees;
```

```
 length
--------
      3
      5
      5
      5
      6
      4
      4
      5
      6
      5
(10 rows)
```

```sql
SELECT * FROM employees WHERE LENGTH(fname) > 5;
```

```
 emp_id | fname  | lname |          email           |  dept   |  salary  | hire_date
--------+--------+-------+--------------------------+---------+----------+------------
      5 | Kavita | Rao   | kavita.rao@example.com   | HR      | 47000.00 | 2020-11-10
      9 | Anjali | Mehta | anjali.mehta@example.com | Finance | 61000.00 | 2018-12-03
(2 rows)
```

1.7 UPPER AND LOWER

```sql
SELECT UPPER(fname) FROM employees;
```

```
 upper
--------
 RAJ
 PRIYA
 ARJUN
 SUMAN
 KAVITA
 AMIT
 NEHA
 RAHUL
 ANJALI
 VIJAY
(10 rows)
```

```sql
SELECT LOWER(fname) FROM employees;
```

```
 lower
--------
 raj
 priya
 arjun
 suman
 kavita
 amit
 neha
 rahul
 anjali
 vijay
(10 rows)
```

1.8 LEFT AND RIGHT

How many characters do you want from left and right

Auto

```
SELECT LEFT('HELLO WORLD', 5);
```

```
 left
-------
 HELLO
(1 row)
```

Auto

```
SELECT RIGHT('HELLO WORLD', 5);
```

```
 right
-------
 WORLD
(1 row)
```

1.9 TRIM

SQL

```
SELECT TRIM('   ALright   ');
```

```
  btrim
---------
 Alright
(1 row)
```

1.10 POSITION

find out the position of a particular character in a word

SQL

```
SELECT POSITION('om' in 'Thomas');
```

```
 position
----------
        3
(1 row)
```

- Practice Exercises

SQL

```
SELECT CONCAT_WS(':', emp_id, fname, lname, dept) FROM employees
WHERE emp_id = 1;
```

```
    concat_ws
-----------------
 1:Raj:Sharma:IT
(1 row)
```

SQL

```
SELECT CONCAT_WS(':', emp_id, CONCAT_WS(' ', fname, lname), dept) FROM
employees
WHERE emp_id = 1;
```

```
    concat_ws
-----------------
 1:Raj Sharma:IT
(1 row)
```

SQL

```sql
SELECT CONCAT_WS(':', emp_id, fname, UPPER(dept)) FROM employees
WHERE emp_id = 4;
```

```
    concat_ws
-----------------
 4:Suman:FINANCE
(1 row)
```

SQL

```sql
SELECT CONCAT(LEFT(dept,1), emp_id), fname from employees;
```

```
 concat | fname
--------+--------
 I1     | Raj
 H2     | Priya
 I3     | Arjun
 F4     | Suman
 H5     | Kavita
 M6     | Amit
 I7     | Neha
 I8     | Rahul
 F9     | Anjali
 M10    | Vijay
(10 rows)
```

→ Find different types of department in databases

```sql
SELECT DISTINCT dept FROM employees;
```

→ Display records from high to low salary

```sql
SELECT * FROM employees ORDER BY salary DESC;
```

→ See only top 3 records from the table

```sql
SELECT * FROM employees LIMIT 3;
```

→ Show the records where first name starts with letter 'A'

```sql
SELECT * FROM employees WHERE fname LIKE 'A%';
```

→ Show record where length of the lname is 4 character

```sql
SELECT * FROM employees WHERE LENGTH(lname) = 4;
```

→ Find the total no of employees in database

```sql
SELECT COUNT(emp_id) FROM employees;
```

→ Find no of employees in each department

```sql
SELECT dept, COUNT(emp_id) FROM employees GROUP BY dept;
```

→ Find the lowest salary paying

```sql
SELECT MIN(salary) FROM employees;
```

→ Find the highest salary paying

```sql
SELECT MAX(salary) FROM employees;
```

→ Find total salary paying in HR dept

```sql
SELECT SUM(SALARY) FROM employees WHERE dept = 'HR';
```

→ AVG salary in each department

```sql
SELECT dept, AVG(salary) FROM employees GROUP BY dept;
```

→ Give me all the data related to a employee who has max salary

```sql
SELECT * FROM employees
WHERE salary = (SELECT MAX(salary) FROM employees);
```

- ALTER QUERY

1.1 ALTER TABLE

→ How to add or remove a column from a table

```sql
ALTER TABLE person
ADD COLUMN age INT;
```

	id integer	name character varying (100)	city character varying (100)	age integer
1	102	Sham	Mumbai	[null]
2	103	Paul	Chennai	[null]
3	101	Raju	Banglore	[null]

SQL

```sql
ALTER TABLE person
DROP COLUMN age;
```

	id integer	name character varying (100)	city character varying (100)
1	102	Sham	Mumbai
2	103	Paul	Chennai
3	101	Raju	Banglore

SQL

```sql
ALTER TABLE person
ADD COLUMN age INT DEFAULT 0;
```

	id integer	name character varying (100)	city character varying (100)	age integer
1	102	Sham	Mumbai	0
2	103	Paul	Chennai	0
3	101	Raju	Banglore	0

→ How to rename a column or table name.

SQL

```sql
ALTER TABLE person
```

```
RENAME COLUMN name TO fname;
```

	id integer	fname character varying (100)	city character varying (100)
1	102	Sham	Mumbai
2	103	Paul	Chennai
3	101	Raju	Banglore

SQL

```
ALTER TABLE person
RENAME TO person_table;

RENAME TABLE person TO person_table;
```

→ How to modify a column

Change the datatype of a column in a table

SQL

```
ALTER TABLE person
ALTER COLUMN fname
SET DATA TYPE VARCHAR(150);
```

	id integer	fname character varying (150)	city character varying (100)
1	102	Sham	Mumbai
2	103	Paul	Chennai
3	101	Raju	Banglore

set the default value for a particular column which is already created.

SQL

```
ALTER TABLE person
ALTER COLUMN fname
SET DEFAULT 'unknown';
```

```
persondb=# \d person;
                          Table "public.person"
 Column |          Type          | Collation | Nullable |           Default
--------+------------------------+-----------+----------+------------------------------
 id     | integer                |           |          |
 fname  | character varying(150) |           |          | 'unknown'::character varying
 city   | character varying(100) |           |          |
```

How to set a column to NOT NULL

```sql
ALTER TABLE person
ALTER COLUMN fname
SET NOT NULL;
```

```
                          Table "public.person"
 Column |          Type          | Collation | Nullable |           Default
--------+------------------------+-----------+----------+------------------------------
 id     | integer                |           |          |
 fname  | character varying(150) |           | not null | 'unknown'::character varying
 city   | character varying(100) |           |          |
```

Drop default value for a particular column

```sql
ALTER TABLE person
ALTER COLUMN fname
DROP DEFAULT;
```

```
                          Table "public.person"
 Column |          Type          | Collation | Nullable | Default
--------+------------------------+-----------+----------+---------
 id     | integer                |           |          |
 fname  | character varying(150) |           | not null |
 city   | character varying(100) |           |          |
```

- CONSTRAINT CHECK

→ We want to make sure the phone no is alteast of 10 digits.

SQL

```sql
ALTER TABLE person
ADD COLUMN mob VARCHAR(15) UNIQUE CHECK (LENGTH(mob) >= 10);
```

```
                    Table "public.person"
 Column |          Type          | Collation | Nullable | Default
--------+------------------------+-----------+----------+---------
 id     | integer                |           |          |
 fname  | character varying(150) |           |          |
 city   | character varying(100) |           |          |
 mob    | character varying(15)  |           |          |
Indexes:
    "person_mob_key" UNIQUE CONSTRAINT, btree (mob)
Check constraints:
    "person_mob_check" CHECK (length(mob::text) >= 10)
```

SQL

```sql
INSERT INTO person(mob)
VALUES(123);
```

```
ERROR:  Failing row contains (null, null, null, 123).new row for relation "person" violates check constraint
"person_mob_check"

ERROR:  new row for relation "person" violates check constraint "person_mob_check"
SQL state: 23514
Detail: Failing row contains (null, null, null, 123).
```

SQL

```sql
INSERT INTO person(mob)
VALUES(1234567890);
```

```
INSERT 0 1

Query returned successfully in 237 msec.
```

→ Drop a particular constraint

SQL

```sql
ALTER TABLE person
DROP CONSTRAINT person_mob_check;
```

```
                    Table "public.person"
 Column |          Type          | Collation | Nullable | Default
--------+------------------------+-----------+----------+---------
 id     | integer                |           |          |
 fname  | character varying(150) |           |          |
 city   | character varying(100) |           |          |
 mob    | character varying(15)  |           |          |
Indexes:
    "person_mob_unique" UNIQUE CONSTRAINT, btree (mob)
```

→ Add a particular constraint

SQL

```sql
ALTER TABLE person
ADD CONSTRAINT mob_not_null CHECK (mob != NULL),
ADD CONSTRAINT person_mob_check CHECK (LENGTH(mob) = 10);
```

```
                    Table "public.person"
 Column |          Type          | Collation | Nullable | Default
--------+------------------------+-----------+----------+---------
 id     | integer                |           |          |
 fname  | character varying(150) |           |          |
 city   | character varying(100) |           |          |
 mob    | character varying(15)  |           |          |
Indexes:
    "person_mob_unique" UNIQUE CONSTRAINT, btree (mob)
Check constraints:
    "mob_not_null" CHECK (mob::text <> NULL::text)
    "person_mob_check" CHECK (length(mob::text) = 10)
```

→ Named Constraint

Auto (SQL)

```sql
CREATE TABLE contacts(
    name VARCHAR(50),
    mob VARCHAR(15) UNIQUE CONSTRAINT mob_less_than_10 CHECK (LENGTH(mob) =
10)
);


ALTER TABLE person
ADD CONSTRAINT mob_less_than_10digits CHECK (LENGTH(mob) >= 10);
```

- Expression CASE

SQL

```sql
SELECT fname, salary,
CASE
    WHEN salary >= 50000 THEN 'HIGH'
    ELSE 'LOW'
END AS salary_category
FROM employees;
```

	fname character varying (100)	salary numeric (10,2)	salary_category text
1	Raj	50000.00	HIGH
2	Priya	45000.00	LOW
3	Arjun	55000.00	HIGH
4	Suman	60000.00	HIGH
5	Kavita	47000.00	LOW
6	Amit	52000.00	HIGH
7	Neha	48000.00	LOW
8	Rahul	53000.00	HIGH
9	Anjali	61000.00	HIGH
10	Vijay	50000.00	HIGH

Multiple conditions

SQL

```sql
SELECT fname, salary,
```

```
CASE
    WHEN salary >= 50000 THEN 'HIGH'
    WHEN salary >= 40000 AND salary < 50000 THEN 'MID'
    ELSE 'LOW'
END AS salary_category
FROM employees;
```

	fname character varying (100)	salary numeric (10,2)	salary_category text
1	Raj	50000.00	HIGH
2	Priya	45000.00	MID
3	Arjun	55000.00	HIGH
4	Suman	60000.00	HIGH
5	Kavita	47000.00	MID
6	Amit	52000.00	HIGH
7	Neha	48000.00	MID
8	Rahul	53000.00	HIGH
9	Anjali	61000.00	HIGH
10	Vijay	50000.00	HIGH

TASK

SQL

```
SELECT fname, salary,
CASE
    WHEN salary > 0 THEN ROUND(salary * 0.10)
END AS bonus
FROM employees;
```

	fname character varying (100)	salary numeric (10,2)	bonus numeric
1	Raj	50000.00	5000
2	Priya	45000.00	4500
3	Arjun	55000.00	5500
4	Suman	60000.00	6000
5	Kavita	47000.00	4700
6	Amit	52000.00	5200
7	Neha	48000.00	4800
8	Rahul	53000.00	5300
9	Anjali	61000.00	6100
10	Vijay	50000.00	5000

SQL

```sql
SELECT
CASE
    WHEN salary >= 50000 THEN 'HIGH'
    WHEN salary >= 48000 AND salary < 50000 THEN 'MID'
    ELSE 'LOW'
END AS salary_category, COUNT(emp_id)
FROM employees GROUP BY salary_category;
```

	salary_category text	count bigint
1	MID	1
2	HIGH	7
3	LOW	2

- Understand RELATIONSHIP

Tables connected to each other

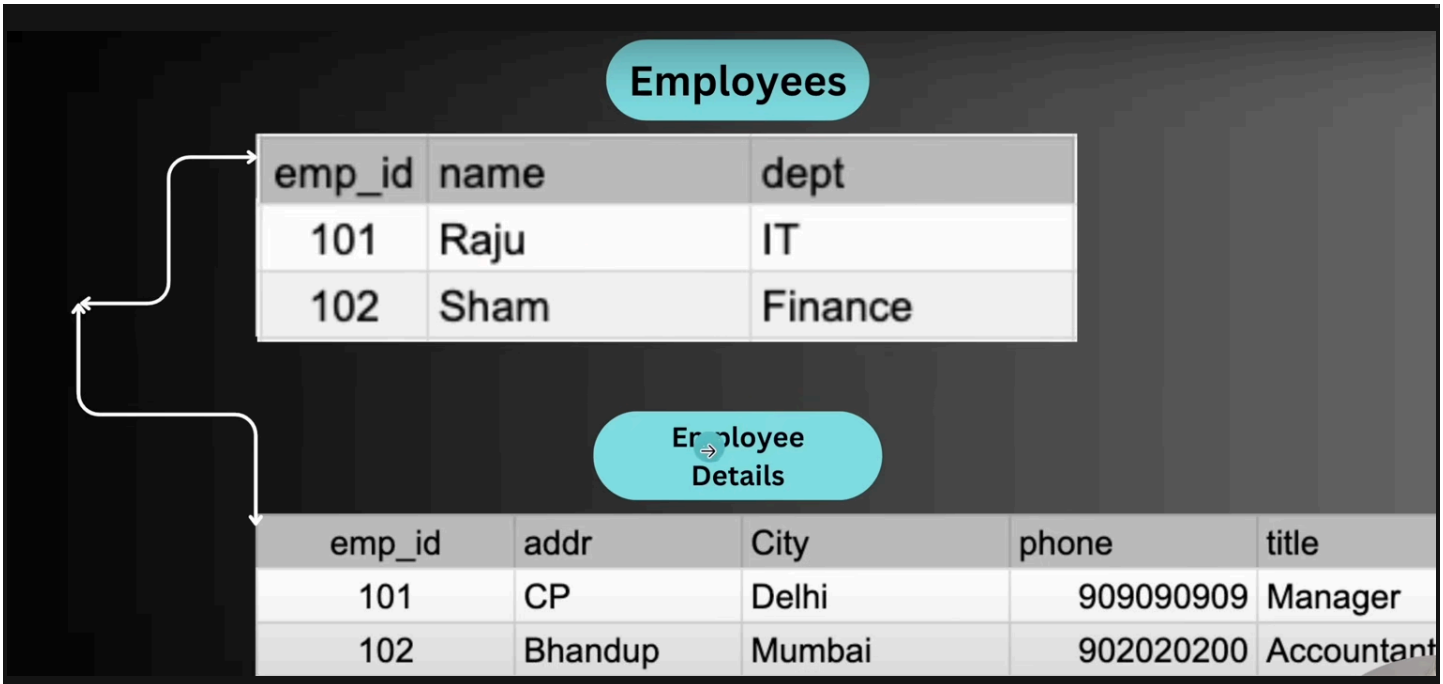
→ Types of Relationship

1.1 One to One

1.2 One to Many

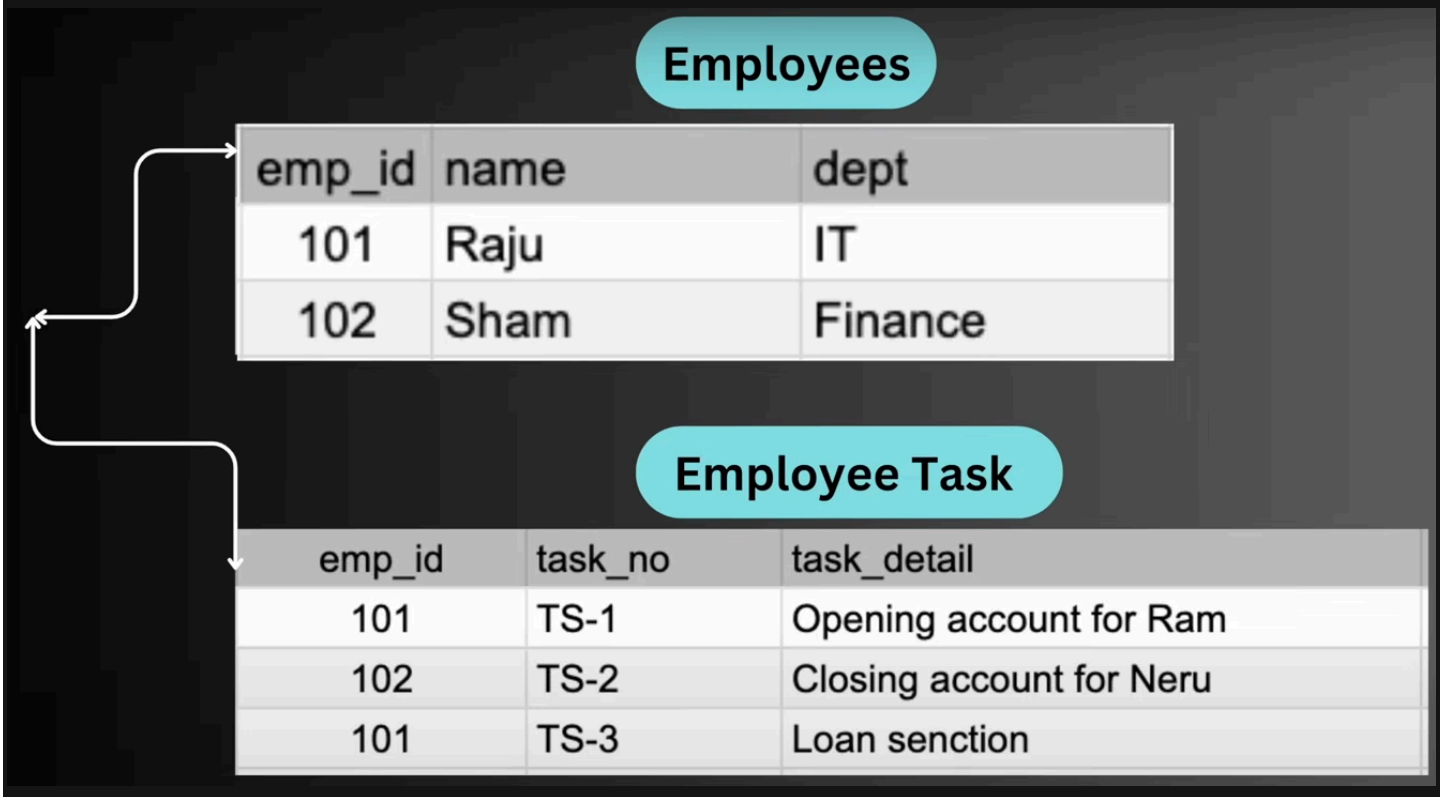
1.3 Many to Many

→ One to One (1:1)



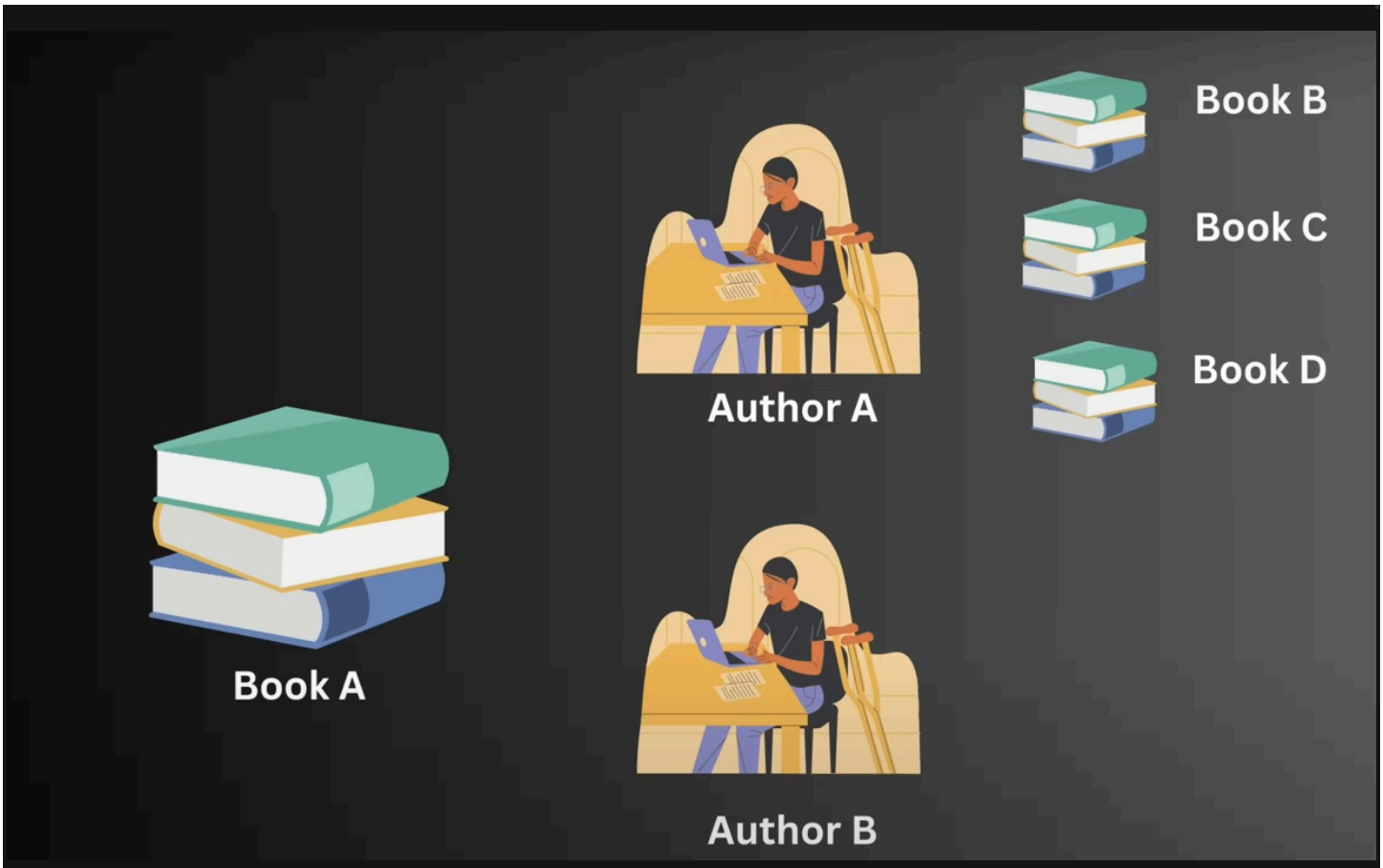
For every employee in employees table will have exactly one personal detail(row) in employee details table.

→ One to Many (1:Many)



Every single employee in employees table can have multiple task completed by him/her in employee task table

→ Many to Many



One book can be written by multiple authors, and one author can also write multiple books.

→ Use case of one to many(1:Many)

Suppose we have a Store and we want to store following data

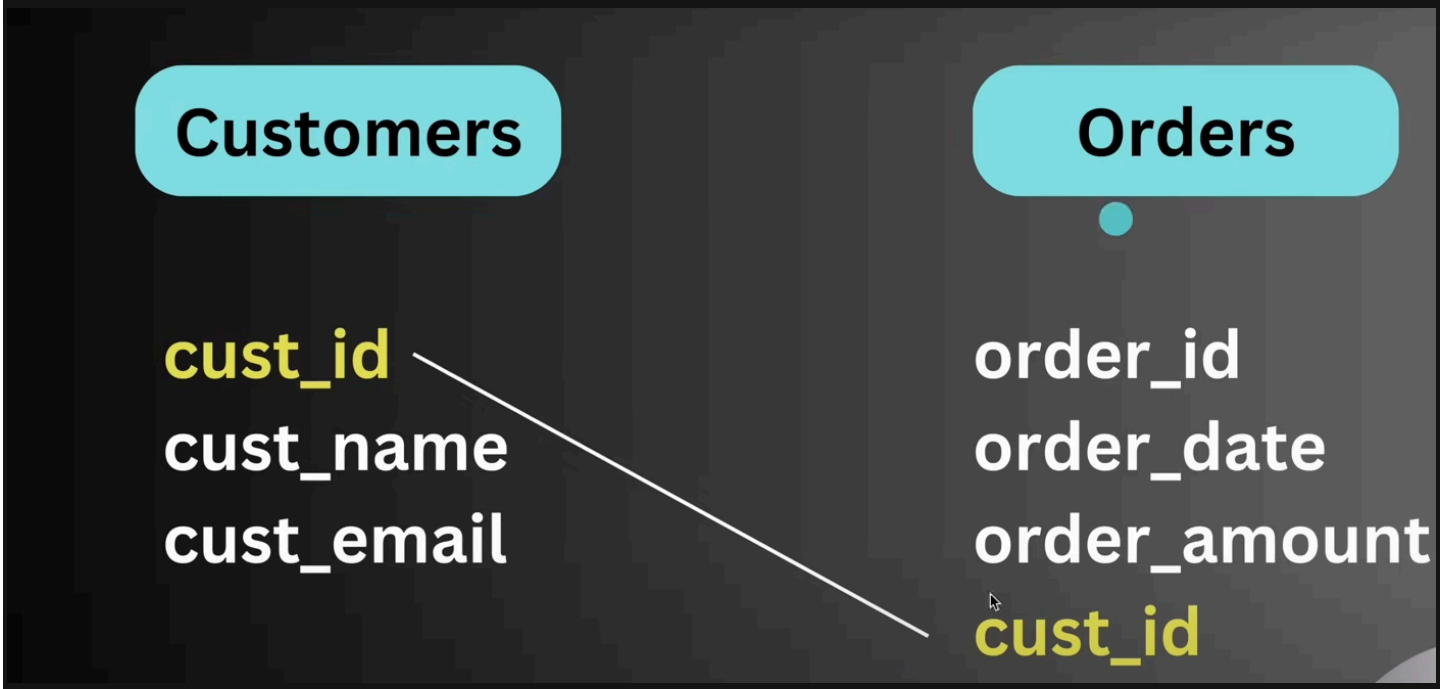
1. Customer name
2. Customer email
3. Order date
4. Order Price

First way to store data

cust_name	email	order_daate	amount
Raju	raju@email.com	2023-05-15	200
Sham	sham@email.com	2023-04-28	500
Raju	raju@email.com	2023-05-14	1000
Baburao	babu@email.com	NULL	NULL
Sham	sham@email.com	2023-03-15	800

Issue → Not Very organized due to Data Redundancy.

Second way to store data (Efficient)



Whenever customer creates an account we register that customer in customers table with cust_id, cust_name, cust_email then when the customer order anything that data will go in orders table with order_id, order_date, order_amount and cust_id(to relate that particular order to specific customer)

Customers

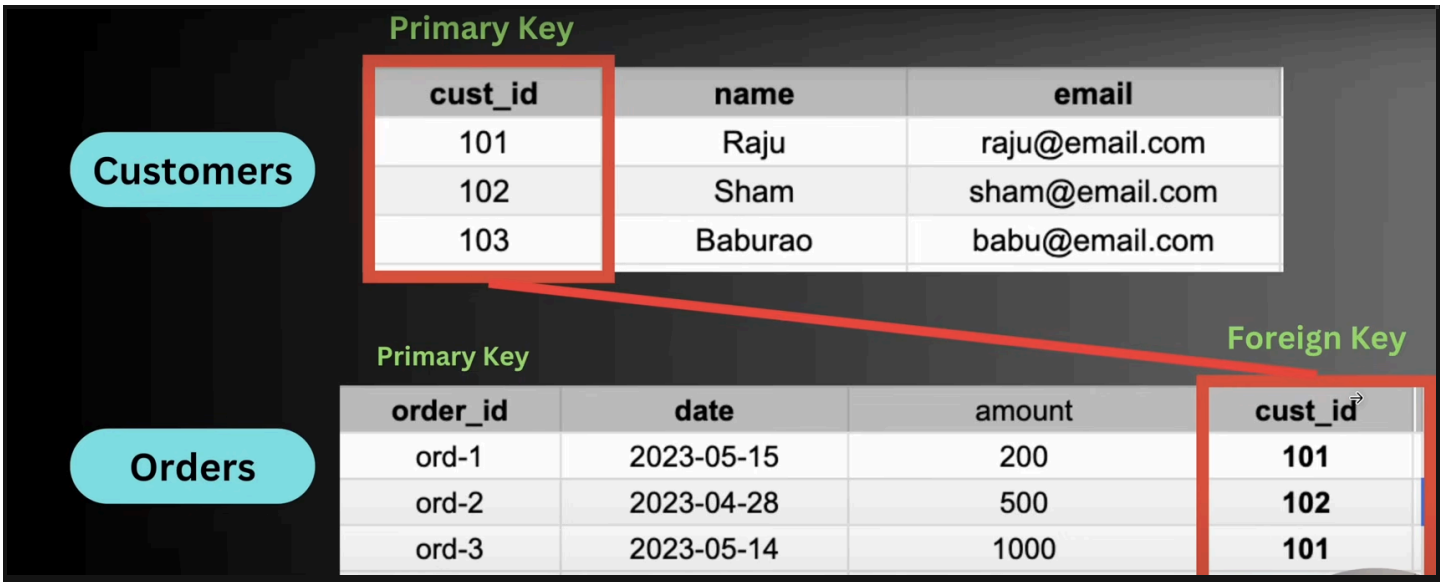
cust_id	name	email
101	Raju	raju@email.com
102	Sham	sham@email.com
103	Baburao	babu@email.com

Orders

order_id	date	amount	cust_id
ord-1	2023-05-15	200	101
ord-2	2023-04-28	500	102
ord-3	2023-05-14	1000	101

This is how we can organize data.

- FOREIGN KEY



a column or columns in a table that references the primary key of another table

- Practical Example of Foreign Key

Create a database called storedb. And create two table customers and orders.

```sql
CREATE TABLE customers(
        cust_id SERIAL PRIMARY KEY,
        cust_name VARCHAR(100) NOT NULL
);

CREATE TABLE orders(
        ord_id SERIAL PRIMARY KEY,
        ord_date DATE NOT NULL,
        price NUMERIC NOT NULL,
        cust_id INT NOT NULL,
        FOREIGN KEY(cust_id) REFERENCES customers(cust_id)
);
```

Insert data

```sql
INSERT INTO customers(cust_name)
VALUES
        ('Raju'),
        ('Sham'),
        ('Paul'),
        ('Alex');

INSERT INTO orders(ord_date, price, cust_id)
VALUES
```

```
('2024-01-01', 250.00, 1),
('2024-01-15', 300.00, 1),
('2024-02-01', 150.00, 2),
('2024-03-01', 450.00, 3),
('2024-04-04', 550.00, 2);
```

```
storedb=# SELECT * FROM customers;
 cust_id | cust_name
---------+-----------
       1 | Raju
       2 | Sham
       3 | Paul
       4 | Alex
(4 rows)


storedb=# SELECT * FROM orders;
 ord_id |  ord_date  | price  | cust_id
--------+------------+--------+---------
      1 | 2024-01-01 | 250.00 |       1
      2 | 2024-01-15 | 300.00 |       1
      3 | 2024-02-01 | 150.00 |       2
      4 | 2024-03-01 | 450.00 |       3
      5 | 2024-04-04 | 550.00 |       2
(5 rows)
```

- Understand JOINS

JOIN operation is used to combine rows from two or more tables based on a related column between them.

→ Types of Join

1. Cross Join
2. Inner Join
3. Left Join
4. Right Join

1. Cross Join

Every row from one table is combined with every row from another table

SQL

```
SELECT * FROM customers CROSS JOIN orders;
```

```
 cust_id | cust_name | ord_id |  ord_date  | price  | cust_id
---------+-----------+--------+------------+--------+---------
       1 | Raju      |      1 | 2024-01-01 | 250.00 |       1
       2 | Sham      |      1 | 2024-01-01 | 250.00 |       1
       3 | Paul      |      1 | 2024-01-01 | 250.00 |       1
       4 | Alex      |      1 | 2024-01-01 | 250.00 |       1
       1 | Raju      |      2 | 2024-01-15 | 300.00 |       1
       2 | Sham      |      2 | 2024-01-15 | 300.00 |       1
       3 | Paul      |      2 | 2024-01-15 | 300.00 |       1
       4 | Alex      |      2 | 2024-01-15 | 300.00 |       1
       1 | Raju      |      3 | 2024-02-01 | 150.00 |       2
       2 | Sham      |      3 | 2024-02-01 | 150.00 |       2
       3 | Paul      |      3 | 2024-02-01 | 150.00 |       2
       4 | Alex      |      3 | 2024-02-01 | 150.00 |       2
       1 | Raju      |      4 | 2024-03-01 | 450.00 |       3
       2 | Sham      |      4 | 2024-03-01 | 450.00 |       3
       3 | Paul      |      4 | 2024-03-01 | 450.00 |       3
       4 | Alex      |      4 | 2024-03-01 | 450.00 |       3
       1 | Raju      |      5 | 2024-04-04 | 550.00 |       2
       2 | Sham      |      5 | 2024-04-04 | 550.00 |       2
       3 | Paul      |      5 | 2024-04-04 | 550.00 |       2
       4 | Alex      |      5 | 2024-04-04 | 550.00 |       2
(20 rows)
```

2. Inner Join

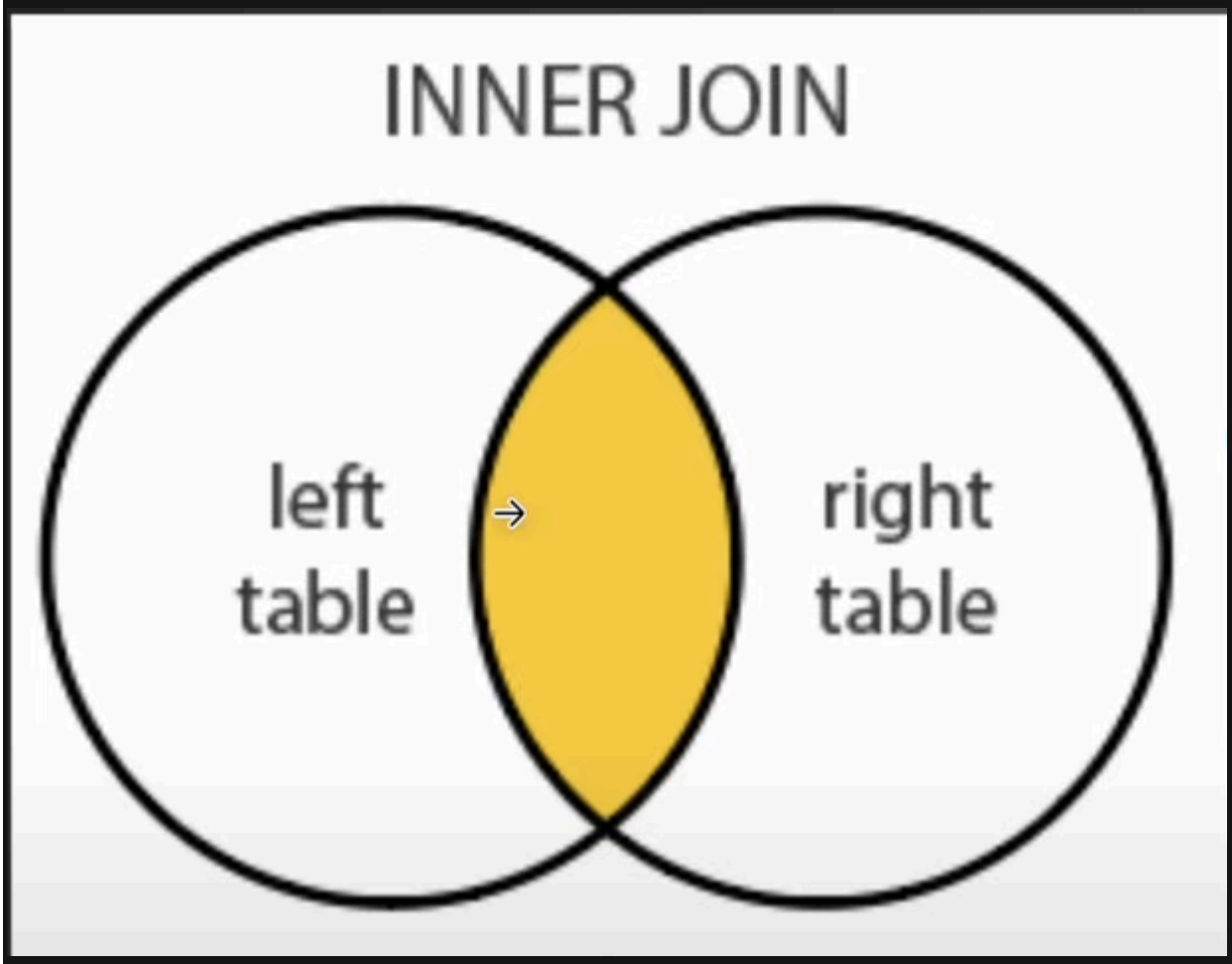
Returns only rows where there is a match between the specified columns in both left and right tables.

SQL

```sql
SELECT * FROM customers c
INNER JOIN
orders o
ON c.cust_id=o.cust_id;
```

cust_id	cust_name	ord_id	ord_date	price	cust_id
1	Raju	1	2024-01-01	250.00	1
1	Raju	2	2024-01-15	300.00	1
2	Sham	3	2024-02-01	150.00	2
3	Paul	4	2024-03-01	450.00	3
2	Sham	5	2024-04-04	550.00	2

(5 rows)



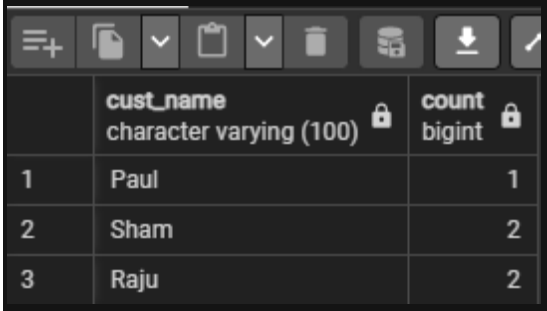
But here we also have customer alex which is not in above table using inner join because he hasnt placed any order so there is no matching column to it in second table.

Inner Join with Group By

How many order did each customer placed.

SQL

```sql
SELECT c.cust_name, COUNT(o.ord_id) FROM customers c
INNER JOIN
orders o
ON c.cust_id=o.cust_id
GROUP BY cust_name;
```



	cust_name character varying (100)	count bigint
1	Paul	1
2	Sham	2
3	Raju	2

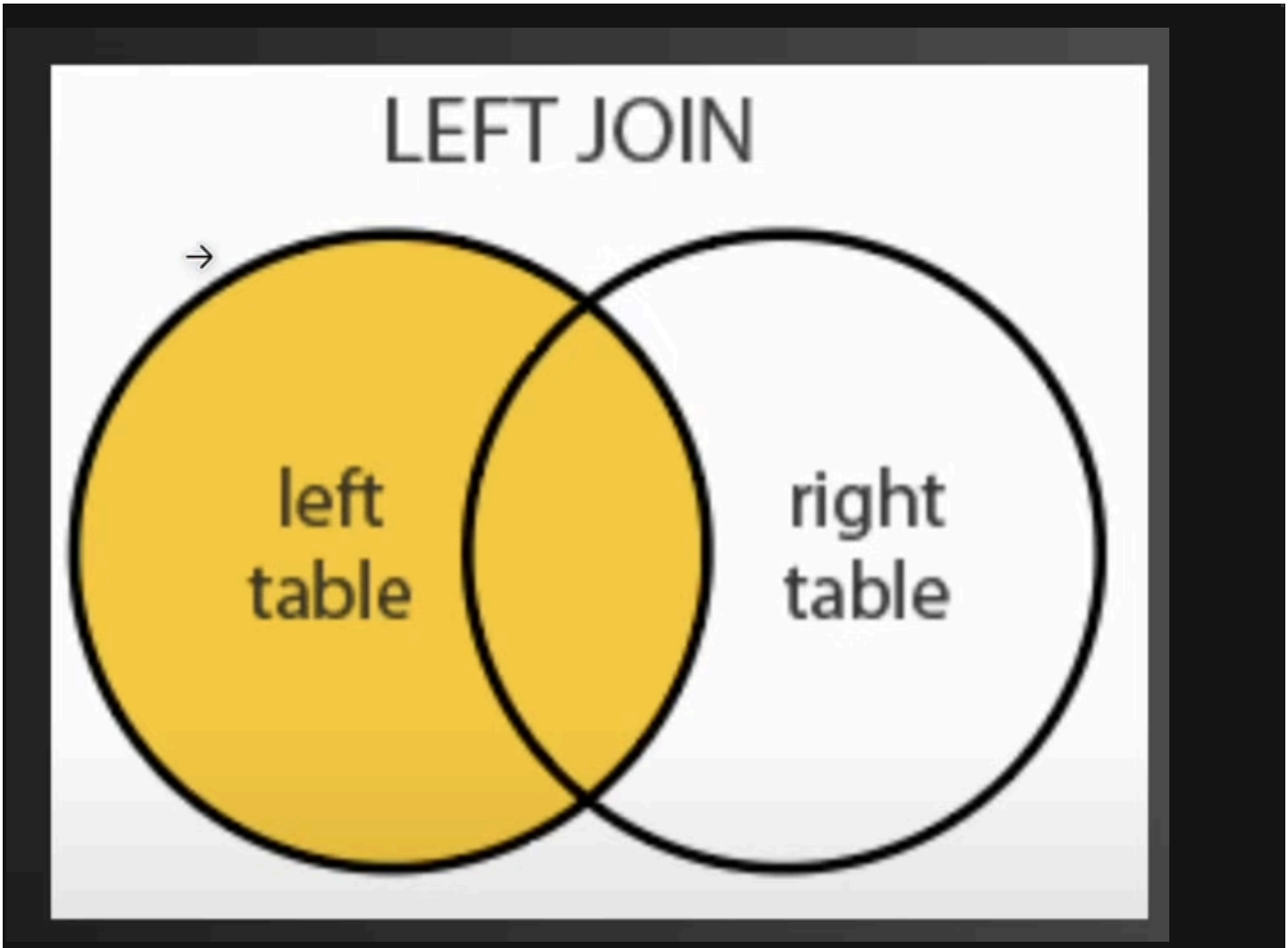
How much did each customer spend

```sql
SELECT c.cust_name, SUM(o.price) FROM customers c
INNER JOIN
orders o
ON c.cust_id=o.cust_id
GROUP BY cust_name;
```

	cust_name character varying (100)	sum numeric
1	Paul	450.00
2	Sham	700.00
3	Raju	550.00

3. Left Join

Returns all rows from left table and the matching rows from the right table.



SQL

```sql
SELECT * FROM customers c
LEFT JOIN
orders o
ON c.cust_id=o.cust_id;
```

	cust_id integer	cust_name character varying (100)	ord_id integer	ord_date date	price numeric	cust_id integer
1	1	Raju	1	2024-01-01	250.00	1
2	1	Raju	2	2024-01-15	300.00	1
3	2	Sham	3	2024-02-01	150.00	2
4	3	Paul	4	2024-03-01	450.00	3
5	2	Sham	5	2024-04-04	550.00	2
6	4	Alex	[null]	[null]	[null]	[null]

4. Right Join

Returns all rows from right table and the matching rows from the left table.

```sql
SELECT * FROM orders o
RIGHT JOIN
customers c
ON c.cust_id=o.cust_id;
```

	ord_id integer	ord_date date	price numeric	cust_id integer	cust_id integer	cust_name character varying (100)
1	1	2024-01-01	250.00	1	1	Raju
2	2	2024-01-15	300.00	1	1	Raju
3	3	2024-02-01	150.00	2	2	Sham
4	4	2024-03-01	450.00	3	3	Paul
5	5	2024-04-04	550.00	2	2	Sham
6	[null]	[null]	[null]	[null]	4	Alex

- Use Case of MANY - MANY Relationship

Ex- One student can opt for many courses. And one course can be opted by many students.

We will create two tables

Students → id, student_name

courses → id, course_name, fees

student_course → student_id, course_id

Create a database called institute. and create 3 tables

```sql
CREATE TABLE students(
        s_id SERIAL PRIMARY KEY,
        name VARCHAR(100) NOT NULL
);

CREATE TABLE courses(
        c_id SERIAL PRIMARY KEY,
        name VARCHAR(100) NOT NULL,
        fee NUMERIC NOT NULL
);

CREATE TABLE enrollment(
```

```
        enrollment_id SERIAL PRIMARY KEY,
        s_id INT NOT NULL,
        c_id INT NOT NULL,
        enrollment_date DATE NOT NULL,
        FOREIGN KEY(s_id) REFERENCES students(s_id),
        FOREIGN KEY(c_id) REFERENCES courses(c_id)
);
```

insert data

SQL

```sql
INSERT INTO students(name)
VALUES
('Raju'),
('Sham'),
('Alex');

INSERT INTO courses(name, fee)
VALUES
('Mathematics', 500.00),
('Physics', 600.00),
('Chemistry', 700.00);

INSERT INTO enrollment (s_id, c_id, enrollment_date)
VALUES
(1, 1, '2024-01-01'), -- Raju enrolled in Mathematics
(1, 2, '2024-01-15'), -- Raju enrolled in Physics
(2, 1, '2024-02-01'), -- Sham enrolled in Mathematics
(2, 3, '2024-02-15'), -- Sham enrolled in Chemistry
(3, 3, '2024-03-25'); -- Alex enrolled in Chemistry
```

Attach 3 tables

SQL

```sql
SELECT s.name, c.name, e.enrollment_date, c.fee FROM enrollment e
JOIN students s ON e.s_id = s.s_id
JOIN courses c ON c.c_id = e.c_id;
```

	name character varying (100)	name character varying (100)	enrollment_date date	fee numeric
1	Raju	Mathematics	2024-01-01	500.00
2	Raju	Physics	2024-01-15	600.00
3	Sham	Mathematics	2024-02-01	500.00
4	Sham	Chemistry	2024-02-15	700.00
5	Alex	Chemistry	2024-03-25	700.00

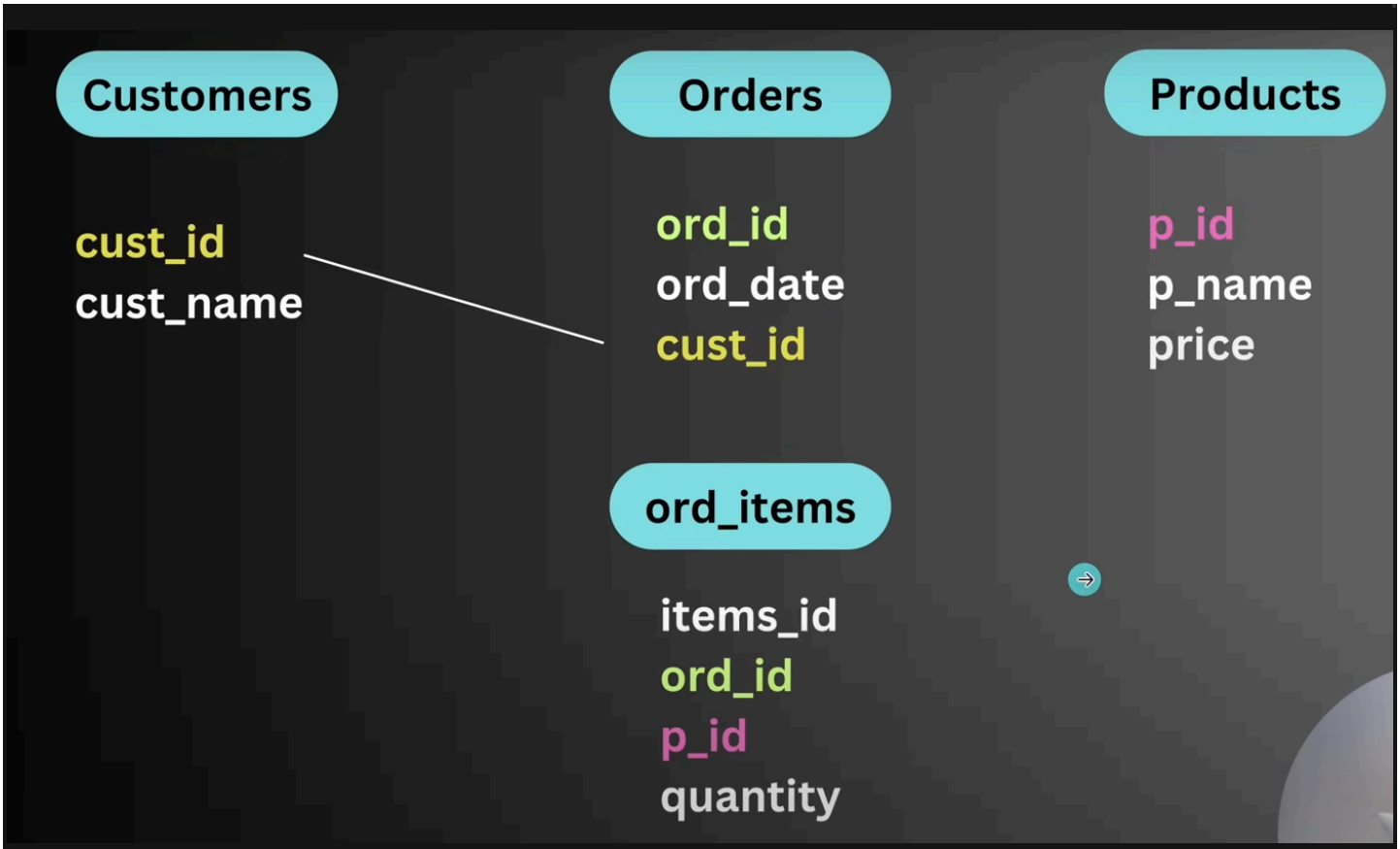
- TASK on RelationShip (Project: E-commerce store)

Question:

Create a one to many and many to many relationship in a shopping store context using four tables.

1. customers
2. orders
3. products
4. order_items

Include a price column in the products table and display the relationship between customers and their orders along with the details of the products in each order.



End Result

cust_name	ord_id	ord_date	p_name	quantity	price	total_price
Sham	4	2024-04-04	Keyboard	1	800.00	800.00
Raju	1	2024-01-01	Laptop	1	55000.00	55000.00
Raju	1	2024-01-01	Cable	2	250.00	500.00
Sham	2	2024-02-01	Laptop	1	55000.00	55000.00
Paul	3	2024-03-01	Mouse	1	500	500
Paul	3	2024-03-01	Cable	3	250.00	750.00
Sham	4	2024-04-04	Keyboard	1	800.00	800.00

Create a database called projectdb and 4 tables as following

SQL

```sql
CREATE TABLE customers (
    cust_id SERIAL PRIMARY KEY,
    cust_name VARCHAR(100) NOT NULL
);

CREATE TABLE orders (
    ord_id SERIAL PRIMARY KEY,
    ord_date DATE NOT NULL,
    cust_id INTEGER NOT NULL,
    FOREIGN KEY (cust_id) REFERENCES customers(cust_id)
);
```

```sql
CREATE TABLE products (
    p_id SERIAL PRIMARY KEY,
    p_name VARCHAR(100) NOT NULL,
    price NUMERIC NOT NULL
);

CREATE TABLE billing (
    item_id SERIAL PRIMARY KEY,
    ord_id INTEGER NOT NULL,
    p_id INTEGER NOT NULL,
    quantity INTEGER NOT NULL,
    FOREIGN KEY (ord_id) REFERENCES orders(ord_id),
    FOREIGN KEY (p_id) REFERENCES products(p_id)
);
```

Insert data

SQL

```sql
INSERT INTO customers (cust_name)
VALUES
    ('Raju'), ('Sham'), ('Paul'), ('Alex');

INSERT INTO orders (ord_date, cust_id)
VALUES
    ('2024-01-01', 1),  -- Raju first order
    ('2024-02-01', 2),  -- Sham first order
    ('2024-03-01', 3),  -- Paul first order
    ('2024-04-04', 2);  -- Sham second order

INSERT INTO products (p_name, price)
VALUES
    ('Laptop', 55000.00),
    ('Mouse', 500),
    ('Keyboard', 800.00),
    ('Cable', 250.00);

INSERT INTO billing (ord_id, p_id, quantity)
VALUES
    (1, 1, 1),  -- Raju ordered 1 Laptop
    (1, 4, 2),  -- Raju ordered 2 Cables
    (2, 1, 1),  -- Sham ordered 1 Laptop
    (3, 2, 1),  -- Paul ordered 1 Mouse
    (3, 4, 5),  -- Paul ordered 5 Cables
    (4, 3, 1);  -- Sham ordered 1 Keyboard
```

Final billing table

SQL

```sql
SELECT  c.cust_name,
        o.ord_date,
        p.p_name,
        p.price,
        b.quantity,
        (b.quantity * p.price) AS total_price FROM billing b
JOIN products p ON b.p_id = p.p_id
JOIN orders o ON b.ord_id = o.ord_id
JOIN customers c ON o.cust_id = c.cust_id;
```

	cust_name character varying (100)	ord_date date	p_name character varying (100)	price numeric	quantity integer	total_price numeric
1	Raju	2024-01-01	Laptop	55000.00	1	55000.00
2	Raju	2024-01-01	Cable	250.00	2	500.00
3	Sham	2024-02-01	Laptop	55000.00	1	55000.00
4	Paul	2024-03-01	Mouse	500	1	500
5	Paul	2024-03-01	Cable	250.00	5	1250.00
6	Sham	2024-04-04	Keyboard	800.00	1	800.00

- VIEWS

If there is some big query which we might be using again and again we can reduce it using VIEW.

Auto (Visual Basic .NET)

```
CREATE VIEW billing_info AS
SELECT  c.cust_name,
                o.ord_date,
                p.p_name,
                p.price,
                b.quantity,
                (b.quantity * p.price) AS total_price FROM billing b
JOIN products p ON b.p_id = p.p_id
JOIN orders o ON b.ord_id = o.ord_id
JOIN customers c ON o.cust_id = c.cust_id;
```

SQL

```
SELECT * FROM billing_info;
```

	cust_name character varying (100)	ord_date date	p_name character varying (100)	price numeric	quantity integer	total_price numeric
1	Raju	2024-01-01	Laptop	55000.00	1	55000.00
2	Raju	2024-01-01	Cable	250.00	2	500.00
3	Sham	2024-02-01	Laptop	55000.00	1	55000.00
4	Paul	2024-03-01	Mouse	500	1	500
5	Paul	2024-03-01	Cable	250.00	5	1250.00
6	Sham	2024-04-04	Keyboard	800.00	1	800.00

- HAVING clause

How much sale per product ?

SQL

```
SELECT p_name, SUM(total_price) FROM billing_info
GROUP BY p_name;
```

	p_name character varying (100)	sum numeric
1	Cable	1750.00
2	Mouse	500
3	Keyboard	800.00
4	Laptop	110000.00

Now show me only products whose sale is above 1500. Where clause doesnt work with GROUP BY we need to use HAVING.

SQL

```
SELECT p_name, SUM(total_price) FROM billing_info
GROUP BY p_name
HAVING SUM(total_price) > 1000;
```

	p_name character varying (100)	sum numeric
1	Cable	1750.00
2	Laptop	110000.00

- GROUP BY ROLLUP

Find the total sum

```sql
SELECT COALESCE(p_name, 'Total'), SUM(total_price) FROM billing_info
GROUP BY ROLLUP(p_name)
ORDER BY SUM(total_price);
```

	coalesce character varying	sum numeric
1	Mouse	500
2	Keyboard	800.00
3	Cable	1750.00
4	Laptop	110000.00
5	Total	113050.00

- Stored Routine

An SQL statement or a set of SQL statement that can be stored on database server which can be call no of times.

Types of Stored routine

1. Stored Procedure
2. User Defined function

1. STORED Procedure

Set of SQL statements that can perform operations such as inserting, updating, deleting and querying data.

Create a procedure for a sql statement where we want to update employee salary.

```sql
CREATE OR REPLACE PROCEDURE update_emp_salary(
    p_employee_id INT,
```

```
    p_new_salary NUMERIC
)
LANGUAGE plpgsql
AS $$
BEGIN
    UPDATE employees
    SET salary = p_new_salary
    WHERE emp_id = p_employee_id;
END;
$$;
```

SQL

```
CALL update_emp_salary(3, 71000);
```

	_id integer	fname character varying (100)	lname character varying (100)	email character varying (100)	dept character varying (50)	salary numeric (10,2)	hire_date date
1	1	Raj	Sharma	raj.sharma@example.com	IT	50000.00	2020-01-15
2	2	Priya	Singh	priya.singh@example.com	HR	45000.00	2019-03-22
3	4	Suman	Patel	suman.patel@example.com	Finance	60000.00	2018-07-30
4	5	Kavita	Rao	kavita.rao@example.com	HR	47000.00	2020-11-10
5	6	Amit	Gupta	amit.gupta@example.com	Marketing	52000.00	2020-09-25
6	7	Neha	Desai	neha.desai@example.com	IT	48000.00	2019-05-18
7	8	Rahul	Kumar	rahul.kumar@example.com	IT	53000.00	2021-02-14
8	9	Anjali	Mehta	anjali.mehta@example.com	Finance	61000.00	2018-12-03
9	10	Vijay	Nair	vijay.nair@example.com	Marketing	50000.00	2020-04-19
10	3	Arjun	Verma	arjun.verma@example.com	IT	71000.00	2021-06-01

Create a procedure for sql statement where we want to add employee.

SQL

```
CREATE OR REPLACE PROCEDURE add_employee(
    p_fname VARCHAR,
    p_lname VARCHAR,
    p_email VARCHAR,
    p_dept VARCHAR,
    p_salary NUMERIC
)
LANGUAGE plpgsql
AS $$
BEGIN
    INSERT INTO employees (fname, lname, email, dept, salary)
    VALUES (p_fname, p_lname, p_email, p_dept, p_salary);
```

```
END;
$$;
```

SQL

```
CALL add_employee('Yash', 'Patil', 'abc@gmail.com', 'IT', 61000)
```

2. User defined Functions

Custom function created by the user to perform specific operations and return a value.

find the name of employees in each department having maximum salary

SQL

```
CREATE OR REPLACE FUNCTION dept_max_sal_emp1(dept_name VARCHAR)
RETURNS TABLE(emp_id INT, fname VARCHAR, salary NUMERIC)
AS $$
BEGIN
    RETURN QUERY
    SELECT
        e.emp_id,  e.fname, e.salary
    FROM
        employees e
    WHERE
        e.dept = dept_name
        AND e.salary = (
            SELECT MAX(emp.salary)
            FROM employees emp
            WHERE emp.dept = dept_name
        );
END;
$$ LANGUAGE plpgsql;
```

SQL

```
SELECT * FROM dept_max_sal_emp1('HR');
```

	emp_id integer	fname character varying	salary numeric
1	5	Kavita	47000.00

- ## Window Function

Window function also known as analytic function allow you to perform calculations across a set of rows related to the current row.

Defined by OVER() clause

Auto (SQL)

```sql
SELECT fname, salary,
       SUM(salary) OVER(ORDER BY salary)
       FROM employees;
```

	fname character varying (100)	salary numeric (10,2)	sum numeric
1	Priya	45000.00	45000.00
2	Kavita	47000.00	92000.00
3	Neha	48000.00	140000.00
4	Raj	50000.00	240000.00
5	Vijay	50000.00	240000.00
6	Amit	52000.00	292000.00
7	Rahul	53000.00	345000.00
8	Suman	60000.00	405000.00
9	Anjali	61000.00	466000.00
10	Arjun	71000.00	537000.00

SQL

```sql
SELECT fname, salary,
       AVG(salary) OVER(ORDER BY salary)
       FROM employees;
```

	fname character varying (100)	salary numeric (10,2)	avg numeric
1	Priya	45000.00	45000.000000000000
2	Kavita	47000.00	46000.000000000000
3	Neha	48000.00	46666.666666666667
4	Raj	50000.00	48000.000000000000
5	Vijay	50000.00	48000.000000000000
6	Amit	52000.00	48666.666666666667
7	Rahul	53000.00	49285.714285714286
8	Suman	60000.00	50625.000000000000
9	Anjali	61000.00	51777.777777777778
10	Arjun	71000.00	53700.000000000000

effective for calculating running totals, moving avgs, rank calculations and cumulative distributions. It does not collapse rows like aggregate functions

SQL

```sql
SELECT
        ROW_NUMBER() OVER(ORDER BY fname),
        fname, salary
        FROM employees;
```

	row_number bigint	fname character varying (100)	salary numeric (10,2)
1	1	Amit	52000.00
2	2	Anjali	61000.00
3	3	Arjun	71000.00
4	4	Kavita	47000.00
5	5	Neha	48000.00
6	6	Priya	45000.00
7	7	Rahul	53000.00
8	8	Raj	50000.00
9	9	Suman	60000.00
10	10	Vijay	50000.00

Auto (SQL)

```sql
SELECT
        ROW_NUMBER() OVER(PARTITION BY dept),
```

```
        fname, dept, salary
        FROM employees;
```

	row_number bigint	fname character varying (100)	dept character varying (50)	salary numeric (10,2)
1	1	Anjali	Finance	61000.00
2	2	Suman	Finance	60000.00
3	1	Priya	HR	45000.00
4	2	Kavita	HR	47000.00
5	1	Arjun	IT	71000.00
6	2	Neha	IT	48000.00
7	3	Rahul	IT	53000.00
8	4	Raj	IT	50000.00
9	1	Vijay	Marketing	50000.00
10	2	Amit	Marketing	52000.00

SQL

```
SELECT
        fname, salary,
        DENSE_RANK() OVER(ORDER BY salary DESC)
        FROM employees;
```

	fname character varying (100)	salary numeric (10,2)	dense_rank bigint
1	Arjun	71000.00	1
2	Anjali	61000.00	2
3	Suman	60000.00	3
4	Rahul	53000.00	4
5	Amit	52000.00	5
6	Raj	50000.00	6
7	Vijay	50000.00	6
8	Neha	48000.00	7
9	Kavita	47000.00	8
10	Priya	45000.00	9

SQL

```sql
SELECT
        fname, salary,
        LAG(salary) OVER()
        FROM employees;
```

	fname character varying (100)	salary numeric (10,2)	lag numeric
1	Raj	50000.00	[null]
2	Priya	45000.00	50000.00
3	Suman	60000.00	45000.00
4	Kavita	47000.00	60000.00
5	Amit	52000.00	47000.00
6	Neha	48000.00	52000.00
7	Rahul	53000.00	48000.00
8	Anjali	61000.00	53000.00
9	Vijay	50000.00	61000.00
10	Arjun	71000.00	50000.00

SQL

```sql
SELECT
        fname, salary,
        LEAD(salary) OVER()
        FROM employees;
```

	fname character varying (100)	salary numeric (10,2)	lead numeric
1	Raj	50000.00	45000.00
2	Priya	45000.00	60000.00
3	Suman	60000.00	47000.00
4	Kavita	47000.00	52000.00
5	Amit	52000.00	48000.00
6	Neha	48000.00	53000.00
7	Rahul	53000.00	61000.00
8	Anjali	61000.00	50000.00
9	Vijay	50000.00	71000.00
10	Arjun	71000.00	[null]

- ## Common Table Expression(CTE)

It is a temporary result set that you can define within a query to simplify complex sql statements.

Use case 1- We want to calculate the average salary per department and then find all the employees whose salary is above the average salary of their department

```SQL
WITH avg_sal AS (
SELECT dept, ROUND(AVG(salary)) AS avg_salary FROM employees GROUP BY dept
)

SELECT
    e.emp_id, e.fname, e.dept, e.salary, a.avg_salary
FROM employees e
JOIN
    avg_sal a ON e.dept = a.dept
WHERE
    e.salary > a.avg_salary;
```

Use case 2 - we want to find the highest paid employee in each department.

```SQL
WITH max_sal AS (
SELECT dept, ROUND(MAX(salary)) AS max_salary FROM employees GROUP BY dept
)

SELECT
```

```
    e.emp_id, e.fname, e.dept, e.salary
FROM employees e
JOIN
    max_sal m ON e.dept = m.dept
WHERE
    e.salary = m.max_salary;
```

	emp_id [PK] integer	fname character varying (100)	dept character varying (50)	salary numeric (10,2)
1	5	Kavita	HR	47000.00
2	6	Amit	Marketing	52000.00
3	9	Anjali	Finance	61000.00
4	3	Arjun	IT	71000.00

Once CTE is created it can only be used once. it will not be persisted.

- Triggers

Triggers are special procedures in a database that automatically execute predefined actions in response to certain events on a specified table or view.

Create a trigger so that if we insert/update negative salary in a table, it will be triggered and set to 0.

Step 2: Create the Trigger Function

sql Copy code

```sql
CREATE OR REPLACE FUNCTION check_salary()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.salary < 0 THEN
        NEW.salary := 0;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Step 3: Create the Trigger

sql Copy code

```sql
CREATE TRIGGER before_insert_salary
BEFORE INSERT ON employees
FOR EACH ROW
EXECUTE FUNCTION check_salary();
```

SQL

```sql
CALL update_emp_salary(2, -52000);
```

	emp_id [PK] integer	salary numeric (10,2)
1	4	60000.00
2	5	47000.00
3	6	52000.00
4	7	48000.00
5	8	53000.00
6	10	50000.00
7	3	71000.00
8	9	71000.00
9	1	-52000.00
10	2	0.00