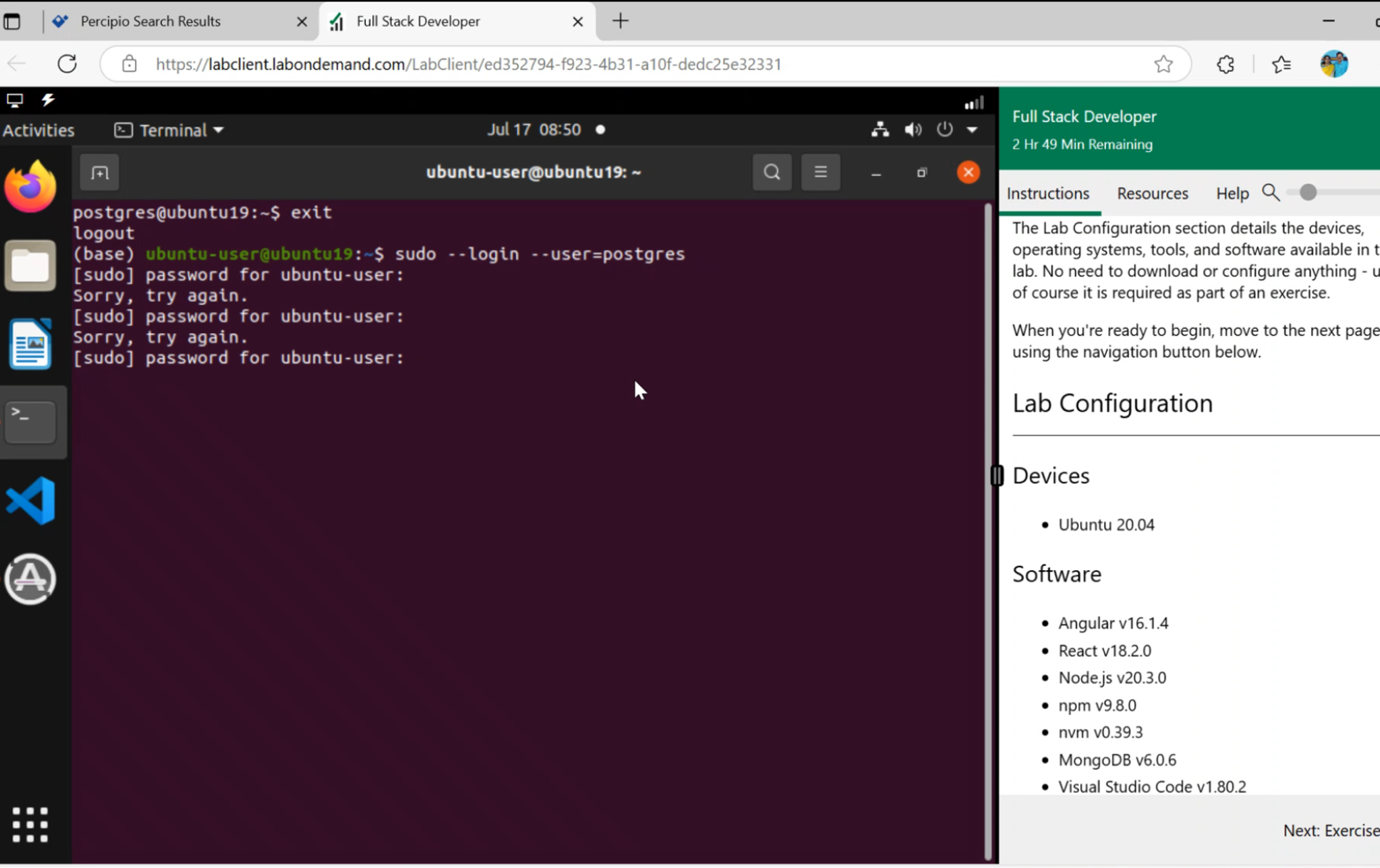
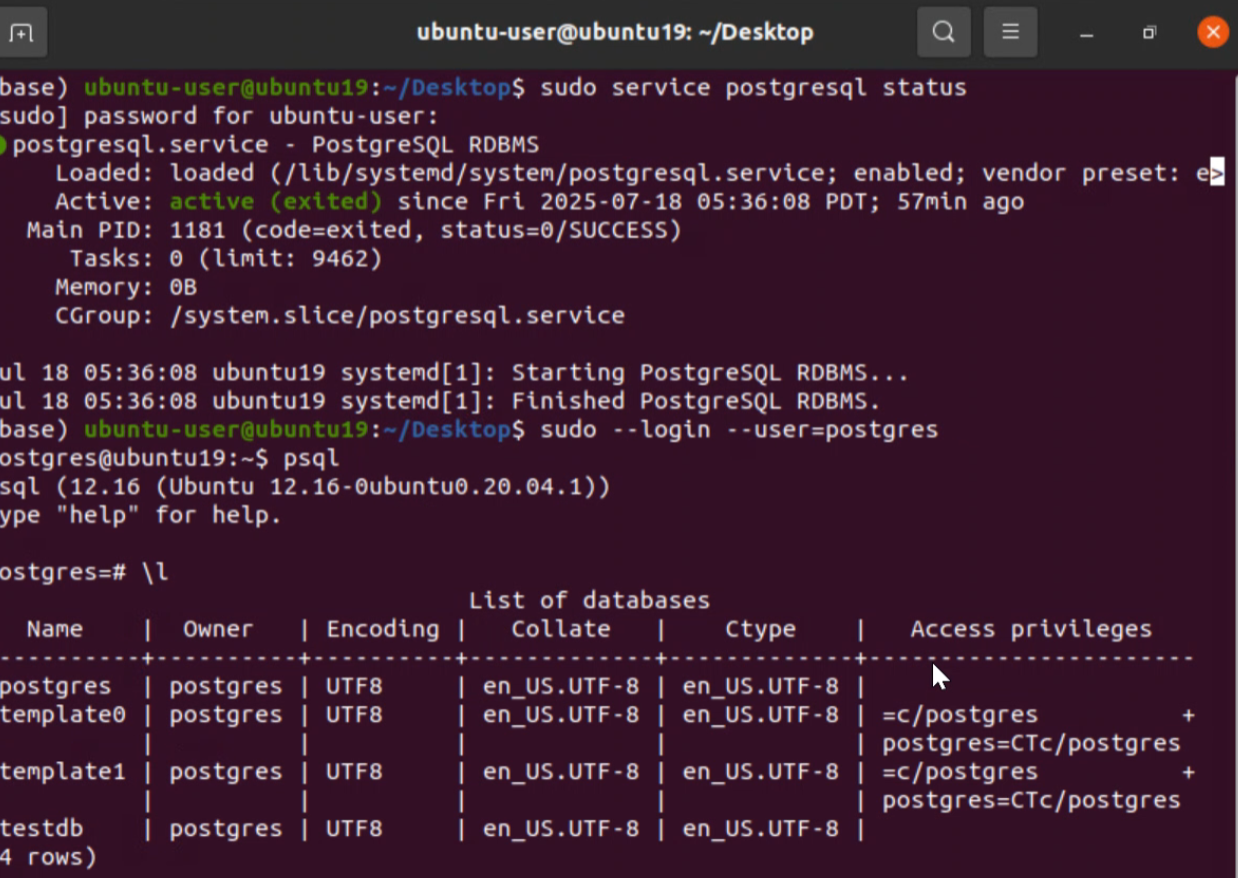
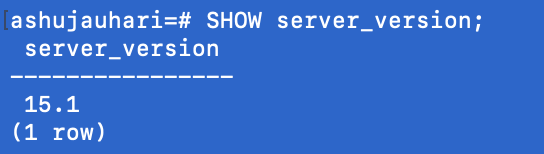
# Day 1- Postgres Script Demo

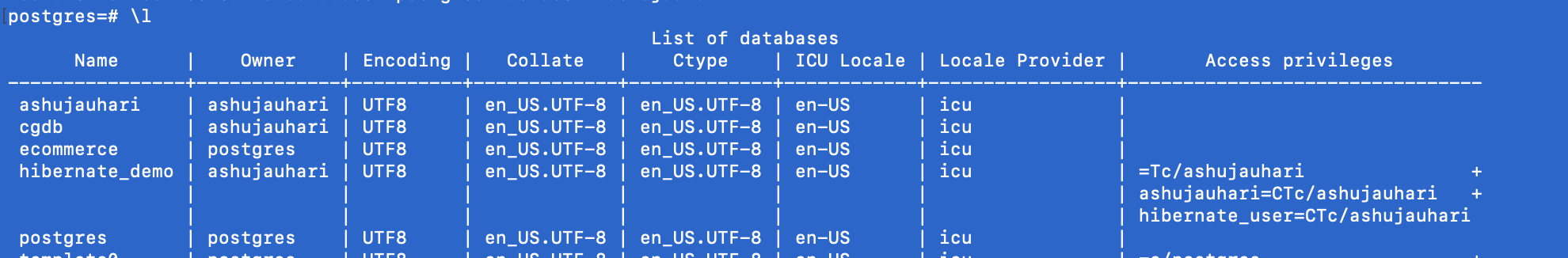
1. 
2. 

Kick Start

1. Start postgres
2. SHOW server\_version;
3. Connect to Database
   1. SELECT current\_database();
   2. Ashujauhari # \c postgres



* 1. postgres=# \l



Create

\! clear

1. Create a Database

postgres=# CREATE DATABASE hr\_system;

# Create Table

CREATE TABLE departments (

dept\_id SERIAL PRIMARY KEY,

dept\_name VARCHAR(50) NOT NULL

);

INSERT INTO departments (dept\_name) VALUES

('Human Resources'),

('Finance'),

('Engineering'),

('Sales'),

('Marketing');

# Insert data to table

INSERT INTO departments (dept\_name) VALUES

('Human Resources'),

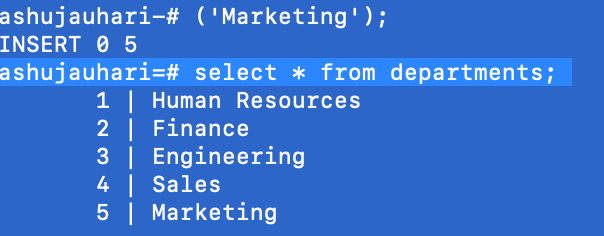
('Finance'),

('Engineering'),

('Sales'),

('Marketing');

1. select \* from departments;



1. CREATE TABLE

CREATE TABLE employees (

emp\_id SERIAL PRIMARY KEY,

emp\_name VARCHAR(100) ,

gender CHAR(1) CHECK (gender IN ('M', 'F')),

hire\_date DATE NOT NULL,

salary NUMERIC(10, 2) NOT NULL CHECK (salary > 0),

dept\_id INT,

FOREIGN KEY (dept\_id) REFERENCES departments(dept\_id)

);

1. SELECT table\_schema, table\_name

FROM information\_schema.tables

WHERE table\_type = 'BASE TABLE'

AND table\_schema NOT IN ('pg\_catalog', 'information\_schema');

INSERT

1. INSERT

Insert INTO employees (emp\_name, gender, hire\_date, salary, dept\_id)

VALUES

('Alice Johnson', 'F', '2021-05-01', 75000, 1),

('Bob Smith', 'M', '2022-01-15', 90000, 2),

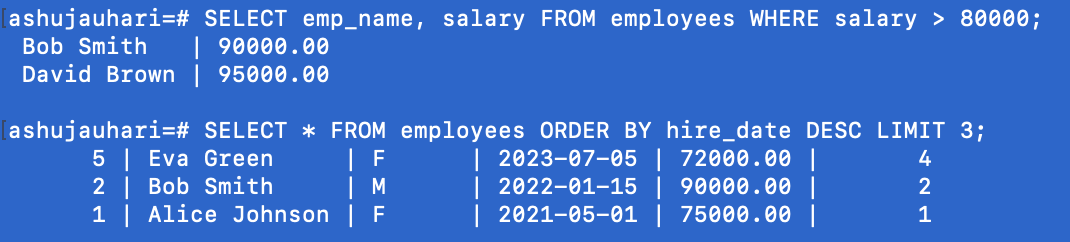
('Catherine Lee', 'F', '2020-09-10', 80000, 3),

('David Brown', 'M', '2019-03-20', 95000, 1),

('Eva Green', 'F', '2023-07-05', 72000, 4);

Select and Where

1. SELECT emp\_name, salary FROM employees WHERE salary > 80000;
2. SELECT \* FROM employees ORDER BY hire\_date DESC LIMIT 3;



Update

1. UPDATE

Update employees SET salary = salary \* 1.1 WHERE dept\_id = 2;

Delete

1. DELETE

Delete FROM employees WHERE emp\_name = 'Charlie Lee';

DELETE FROM employees

WHERE emp\_name LIKE 'Charli%';

Functions

1. SELECT COUNT(\*) AS total\_employees FROM employees;
2. SELECT SUM(salary) AS total\_payroll FROM employee;
3. SELECT AVG(salary) AS average\_salary FROM employees;
4. SELECT emp\_name, salary FROM employees WHERE salary > (SELECT AVG(salary) FROM employees);
5. SELECT e.emp\_name, d.dept\_name FROM employees e current\_date

departments d ON e.dept\_id = d.dept\_id;

1. SELECT MIN(age) AS youngest\_employee FROM employees;
2. SELECT MAX(salary) AS highest\_salary FROM employees;

String Functions

1. SELECT LENGTH('PostgreSQL') AS len;
2. SELECT LOWER('PostgreSQL Rocks!') AS lower\_str;
3. SELECT UPPER('PostgreSQL Rocks!') AS upper\_str;
4. SELECT INITCAP('postgresql rocks!') AS initcap\_str;
5. SELECT CONCAT('Hello', ' ', 'World') AS greeting;
6. SELECT SUBSTRING('PostgreSQL' FROM 1 FOR 4) AS sub\_str;
7. SELECT TRIM(' Hello World ') AS trimmed\_str;
8. SELECT REPLACE('PostgreSQL is awesome', 'awesome', 'powerful') AS replaced\_str;

JOIN

1. --Inner Join--

SELECT e.emp\_name, d.dept\_name

FROM employees e

JOIN departments d ON e.dept\_id = d.dept\_id;

SELECT e.emp\_name, d.dept\_name

FROM employees e

LEFT JOIN departments d ON e.dept\_id = d.dept\_id;

SELECT

e.emp\_name,

e.dept\_id AS emp\_dept\_id,

d.dept\_name,

d.dept\_id AS dept\_dept\_id

FROM

employees e

FULL OUTER JOIN

departments d ON e.dept\_id = d.dept\_id;

Alter Table

1. ALTER T- ADD COLUMN

ALTER TABLE employees

ADD COLUMN manager\_id INTEGER;

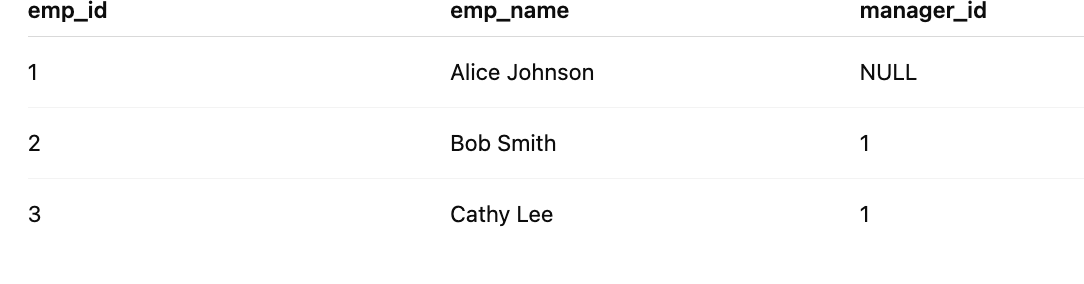
ALTER TABLE employees

ADD COLUMN mobile INTEGER;

1. ALTER TABLE employees RENAME COLUMN phone\_number TO mobile;
2. ALTER TABLE employees DROP COLUMN mobile;
3. ALTER TABLE employees

ADD CONSTRAINT fk\_manager

FOREIGN KEY (manager\_id) REFERENCES employees(emp\_id);



Group BY

1. Group BY

SELECT dept\_id, COUNT(\*) AS employee\_count

FROM employees

GROUP BY dept\_id;

1. SELECT gender, COUNT(\*) AS gender\_count

FROM employees

GROUP BY gender;

1. SELECT dept\_id,

SUM(salary) AS total\_salary,

AVG(salary) AS average\_salary

FROM employees

GROUP BY dept\_id;

SELECT

d.dept\_name,

COUNT(e.emp\_id) AS total\_employees

FROM

employees e

INNER JOIN

departments d ON e.dept\_id = d.dept\_id

GROUP BY

d.dept\_name;

26. Average salary per department

SELECT

d.dept\_name,

AVG(e.salary) AS avg\_salary

FROM

employees e

INNER JOIN

departments d ON e.dept\_id = d.dept\_id

GROUP BY

d.dept\_name;

1. SELECT EXTRACT(YEAR FROM hire\_date) AS hire\_year,

COUNT(\*) AS hires

FROM employees

GROUP BY hire\_year

ORDER BY hire\_year;

View

1. CREATE VIEW contact\_info AS SELECT emp\_id, emp\_name FROM employees;
2. UPDATE contact\_info SET emp\_name = 'Me' WHERE emp\_id = 2;
3. DROP VIEW IF EXISTS dept\_salary\_report;
4. CREATE VIEW contact\_info AS
5. SELECT emp\_id, emp\_name FROM employees;
6. UPDATE contact\_info SET emp\_name = 'Me' WHERE emp\_id = 2;
7. DROP VIEW IF EXISTS dept\_salary\_report;

Index

1. CREATE INDEX idx ON employees(dept\_id);
2. CREATE INDEX idx\_salary ON employees(salary);
3. \di

EXPLAIN ANALYZE

1. EXPLAIN ANALYZE

SELECT e.emp\_name, e.salary, d.dept\_name

FROM employees e

JOIN departments d ON e.dept\_id = d.dept\_id

WHERE e.salary > (

SELECT AVG(salary) FROM employees WHERE dept\_id = e.dept\_id

);

1. EXPLAIN ANALYZE

SELECT d.dept\_name, COUNT(e.emp\_id)

FROM departments d

LEFT JOIN employees e ON d.dept\_id = e.dept\_id

GROUP BY d.dept\_name;

Function

1. SELECT emp\_name, LENGTH(emp\_name) AS name\_length FROM employees;
2. SELECT emp\_name, LOWER(email) AS lowercase\_email FROM employees;
3. SELECT emp\_name, UPPER(emp\_name) AS name\_caps FROM employees;
4. SELECT emp\_name, INITCAP(LOWER(emp\_name)) AS proper\_case FROM employees;
5. SELECT emp\_name, CONCAT(emp\_name, ' <', gender, '>') AS contact\_display FROM employees;
6. SELECT emp\_name, SUBSTRING(emp\_name FROM 1 FOR 5) AS partial\_name FROM employees;
7. SELECT emp\_name, REPLACE(email, '@company.com', '@corp.com') AS updated\_email FROM employees;
8. SELECT emp\_id, REPLACE(emp\_name, 'Eva', 'Jonathan') AS updated\_name correlated FROM employees;

Date Function

1. SELECT CURRENT\_DATE AS today;
2. SELECT CURRENT\_TIMESTAMP AS current\_date\_time;
3. SELECT EXTRACT(YEAR FROM CURRENT\_DATE) AS year,

EXTRACT(MONTH FROM CURRENT\_DATE) AS month,

EXTRACT(DAY FROM CURRENT\_DATE) AS day;

1. SELECT NOW() AS current\_timestamp;
2. SELECT

DATE\_TRUNC('month', CURRENT\_TIMESTAMP) AS month\_start,

DATE\_TRUNC('day', CURRENT\_TIMESTAMP) AS day\_start;

//  month\_start: the timestamp set to the **first day of the current month at 00:00:00**.

 day\_start: the timestamp set to the **start of the current day at 00:00:00**.

1. SELECT TO\_CHAR(CURRENT\_DATE, 'DD-Mon-YYYY') AS formatted\_date;
2. SELECT AGE(DATE '2025-07-17', DATE '2000-01-01') AS age\_between\_dates;

JSONb

1. CREATE TABLE Products (

product\_id SERIAL PRIMARY KEY,

name VARCHAR(100),

attributes JSONB

);

1. INSERT INTO products (name, attributes) VALUES

('Laptop', '{

"brand": "Dell",

"specs": { "ram": "16GB", "storage": "512GB" },

"price": 799.99

}'::jsonb),

('Smartphone', '{

"brand": "Samsung",

"specs": { "ram": "8GB", "storage": "128GB" },

"price": 599.49

}'::jsonb),

('Tablet', '{

"brand": "Apple",

"specs": { "ram": "4GB", "storage": "64GB" },

"price": 429.99

}'::jsonb);

**Get brand: -> retrieves JSON value**

1. SELECT name, attributes -> 'brand' AS brand FROM products;

#### **Get RAM size using -> and ->>: ->> retrieves text value.**

1. SELECT name,attributes -> 'specs' ->> 'ram' AS ram\_size FROM products;

**Use #> for nested keys (as array):**

1. SELECT name, attributes #> '{specs,storage}' AS storage\_size

FROM products;

**Use jsonb\_each() to expand key-value pairs**

1. SELECT name, key, value

FROM products, jsonb\_each(attributes);

**Use jsonb\_array\_elements() (if value is an array)**

1. UPDATE products

SET attributes = jsonb\_set(attributes, '{features}', '["Touchscreen", "Backlit Keyboard"]');

**Filter Records Using JSONB**

1. SELECT \*

FROM products

WHERE (attributes ->> 'price')::numeric > 500;

1. SELECT \*

FROM products

WHERE attributes ? 'brand';

Scalar Subqueries

Returns **a single value (one row, one column)**.

**Example: Get all employees whose salary is above the average salary**

1. SELECT emp\_id, emp\_name, salary

FROM employees

WHERE salary > (SELECT AVG(salary) FROM employees);

**Employees earning above average**

1. SELECT emp\_id, emp\_name, salary

FROM employees

WHERE salary > (SELECT AVG(salary) FROM employees);

**Find Employees with Salary = MAX Salary**

1. SELECT emp\_id, emp\_name, salary

FROM employees

WHERE salary = (SELECT MAX(salary) FROM employees);

**Employees from the Largest Department**

1. Select emp\_id, emp\_name

FROM employees

WHERE dept\_id = (

SELECT dept\_id

FROM employees

GROUP BY dept\_id

ORDER BY COUNT(\*) DESC

LIMIT 1 );

**Department Located in Specific City**

1. SELECT \*

FROM departments

WHERE location = (

SELECT location

FROM departments

WHERE dept\_name = 'Sales');

**Difference Between Max and Min Salary**

1. SELECT (

(SELECT MAX(salary) FROM employees) -

(SELECT MIN(salary) FROM employees)

) AS salary\_gap;

**Show Total Salary of Employees in Smallest Department**

1. SELECT SUM(salary)

FROM employees

WHERE department\_id = (

SELECT department\_id

FROM employees

GROUP BY department\_id

ORDER BY COUNT(\*) ASC

LIMIT 1

);

**Hire Date of the Most Recent Joiner**

1. SELECT name, hire\_date

FROM employees

WHERE hire\_date = (

SELECT MAX(hire\_date) FROM employees);

Correlated Subqueries

## Employees with salary greater than department average

// **Check average salaries per department**:

SELECT e.emp\_name, e.salary, e.dept\_id,

(SELECT AVG(e2.salary)

FROM employees e2

WHERE e2.dept\_id = e.dept\_id) AS dept\_avg

FROM employees e;

INSERT INTO employees (emp\_name, gender, hire\_date, salary, dept\_id) VALUES

('Tom Hill', 'M', '2021-02-10', 70000, 1),

('Amy Wong', 'F', '2021-05-12', 67000, 1),

('Raj Patel', 'M', '2022-11-03', 105000, 2),

('Linda Roe', 'F', '2020-04-22', 78000, 3);

SELECT e.emp\_name, e.salary, e.dept\_id

FROM employees e

WHERE e.salary > (

SELECT AVG(e2.salary)

FROM employees e2

WHERE e2.dept\_id = e.dept\_id

);

SELECT e.emp\_name, e.salary, e.dept\_id

FROM employees e

WHERE EXISTS (

SELECT 1

FROM employees e2

WHERE e2.dept\_id = e.dept\_id

GROUP BY e2.dept\_id

HAVING e.salary > AVG(e2.salary)

);

## Customers who placed orders worth more than their average order value

SELECT c.customer\_id, c.name

FROM customers c

WHERE EXISTS (

SELECT 1

FROM orders o

WHERE o.customer\_id = c.customer\_id

AND o.total > (

SELECT AVG(o2.total)

FROM orders o2

WHERE o2.customer\_id = c.customer\_id

));

## Find employees who earn more than their manager

SELECT e.emp\_name, e.salary

FROM employees e

WHERE e.salary > (

SELECT m.salary

FROM employees m

WHERE m.emp\_id = e.manager\_id

);

**Windows Functions**

**RANK employees by salary** in each department

SELECT emp\_name, dept\_id, salary,

RANK() OVER (PARTITION BY dept\_id ORDER BY salary DESC) AS rank\_in\_dept

FROM employees;

**Calculate average salary** in each department along with individual salary

SELECT emp\_name, dept\_id, salary,

AVG(salary) OVER (PARTITION BY dept\_id) AS avg\_salary\_dept

FROM employees;

**Row number** of each employee based on hire date

SELECT emp\_name, hire\_date,

ROW\_NUMBER() OVER (ORDER BY hire\_date ASC) AS row\_num\_by\_hire\_date

FROM employees;

**Cumulative salary** (running total) by department

SELECT emp\_name, dept\_id, salary,

SUM(salary) OVER (PARTITION BY dept\_id ORDER BY emp\_name) AS running\_total\_salary

FROM employees;

### **Previous employee salary** in the same department (using LAG)

SELECT emp\_name, dept\_id, salary,

LAG(salary, 1) OVER (PARTITION BY dept\_id ORDER BY salary) AS previous\_salary

FROM employees;

**Next employee salary** in the same department (using LEAD)

SELECT emp\_name, dept\_id, salary,

LEAD(salary, 1) OVER (PARTITION BY dept\_id ORDER BY salary) AS next\_salary

FROM employees;

**Find % difference from department average salary**

SELECT emp\_name, dept\_id, salary,

ROUND((salary - AVG(salary) OVER (PARTITION BY dept\_id)) /

AVG(salary) OVER (PARTITION BY dept\_id) \* 100, 2) AS percent\_diff\_from\_avg

FROM employees;

Common Table Expressions (CTEs)

**Get employees with above-average salary using CTE**

WITH avg\_salary\_cte AS (

SELECT AVG(salary) AS avg\_sal

FROM employees

)

SELECT emp\_name, salary

FROM employees, avg\_salary\_cte

WHERE employees.salary > avg\_salary\_cte.avg\_sal;

**Security and Access Control**

CREATE ROLE analyst LOGIN PASSWORD 'securepass';

GRANT CONNECT ON DATABASE companydb TO analyst;

Privileges and GRANT/REVOKE

GRANT SELECT ON employees TO analyst;

REVOKE UPDATE ON employees FROM analyst;

Row-Level Security (RLS)

Enable RLS

ALTER TABLE employees ENABLE ROW LEVEL SECURITY;

-- Create a policy

CREATE POLICY emp\_policy

ON employees

FOR SELECT USING (emp\_id = current\_user::int);

SELECT to\_tsvector('english', 'The quick brown fox jumps over the lazy dog.');

## Step-by-Step: Add tsvector and Full-Text Search

* + 1. Add a search\_vector column:
       - ALTER TABLE employees ADD COLUMN search\_vector tsvector;
    2. Populate search\_vector with searchable fields:

UPDATE employees

SET search\_vector = to\_tsvector(

'english',

emp\_name || ' ' || gender || ' ' || salary::TEXT

);

* + 1. Create a GIN index for fast searching:

CREATE INDEX idx\_gin\_employee\_search

ON employees USING **GIN**(search\_vector);

* + 1. Example full-text query:

-- Search for employees with name or gender containing "M" and "50000"

SELECT emp\_id, emp\_name, gender, salary

FROM employees

WHERE search\_vector @@ to\_tsquery('M & 50000');

Advanced index types (GIN, GiST, BRIN)

# GIST

CREATE TABLE bookings (

id SERIAL,

room\_id INT,

availability DATERANGE

);

INSERT INTO bookings (room\_id, availability)

VALUES (101, '[2025-08-01, 2025-08-10)');

CREATE INDEX idx\_gist1\_availability

ON bookings USING GIST(availability);

CREATE TABLE meetings (

id SERIAL PRIMARY KEY,

room\_id INT,

meeting\_time TSRANGE

);

INSERT INTO meetings (room\_id, meeting\_time)

VALUES (1, '[2025-07-19 10:00, 2025-07-19 11:30)');

INSERT INTO meetings (room\_id, meeting\_time)

VALUES (1, tsrange('2025-07-19 14:00', '2025-07-19 15:00', '[)'));

# BRIN

**Step 1: Table with timestamped sensor data:**

CREATE TABLE sensor\_data1 (

id SERIAL PRIMARY KEY,

sensor\_id INT,

reading NUMERIC,

reading\_time TIMESTAMP

);

Step 2: Create BRIN index.

CREATE INDEX idx\_brin\_senso1r\_time

ON sensor\_data USING BRIN(reading\_time);

INSERT INTO sensor\_data1 (sensor\_id, reading, reading\_time) VALUES

(101, 23.7, '2025-07-19 08:30:00'),

(102, 18.5, '2025-07-19 08:31:15'),

(101, 24.1, '2025-07-19 08:32:45'),

(103, 22.9, '2025-07-19 08:34:00'),

(102, 19.3, '2025-07-19 08:35:30');

Step 3: Query that benefits from BRIN:

SELECT \* FROM sensor\_data

WHERE reading\_time BETWEEN '2025-07-10' AND '2025-07-19';

Basic Backup Command

pg\_dump -U postgres -d mydb > mydb\_backup.sql

-U User name

-d Database name

> Redirect output to file

Backup with Custom Format (Compressed)

pg\_dump -U postgres -F c -f mydb\_backup.dump mydb

| **Flag** | **Description** |
| --- | --- |
| -F c | Format = custom (compressed) |
| -f | Output file name |

Restore

3. Restoring from SQL File

psql -U postgres -d newdb < mydb\_backup.sql

4. Restoring from Custom Dump

pg\_restore -U postgres -d newdb mydb\_backup.dump

You can also **list contents**:

pg\_restore -l mydb\_backup.dump

PITR

Step 1. Take a Physical Base Backup

pg\_basebackup -U postgres -D /var/lib/postgresql/backup/ -Ft -z -P

| **Flag** | **Description** |
| --- | --- |
| -D | Directory to store backup |
| -Ft | Format = tar |
| -z | Compress |
| -P | Show progress |

Step 2. Enable WAL Archiving (in postgresql.conf)

wal\_level = replica

archive\_mode = on

archive\_command = 'cp %p /var/lib/postgresql/wal\_archive/%f'

Step 3: Restore and Recover to Point-in-Time

* 1. Stop PostgreSQL

sudo systemctl stop postgresql

* 1. Replace Data Directory with Backup

rm -rf /var/lib/postgresql/14/main/\*

tar -xvf base.tar -C /var/lib/postgresql/14/main/

* 1. Create recovery.signal and postgresql.auto.conf

touch /var/lib/postgresql/14/main/recovery.signal

echo "restore\_command = 'cp /var/lib/postgresql/wal\_archive/%f %p'" >> postgresql.auto.conf

echo "recovery\_target\_time = '2025-07-18 10:45:00'" >> postgresql.auto.conf

* 1. Start PostgreSQL

sudo systemctl start postgresql

Trigger

CREATE TABLE departments (

dept\_id SERIAL PRIMARY KEY,

dept\_name VARCHAR(100) NOT NULL

);

CREATE TABLE employees (

emp\_id SERIAL PRIMARY KEY,

emp\_name VARCHAR(100) NOT NULL,

gender CHAR(1),

hire\_date DATE NOT NULL,

salary NUMERIC(10,2) NOT NULL,

dept\_id INTEGER REFERENCES departments(dept\_id),

manager\_id INTEGER

);

## Create audit\_log table for trigger logging

CREATE TABLE audit\_log (

log\_id SERIAL PRIMARY KEY,

action\_time TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

action TEXT

);

## Write the Trigger Function

CREATE OR REPLACE FUNCTION log\_new\_employee()

RETURNS TRIGGER AS $$

BEGIN

INSERT INTO audit\_log(action)

VALUES (

'New employee added: ' || NEW.emp\_name ||

', Department ID: ' || NEW.dept\_id

);

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

**Create the Trigger**

CREATE TRIGGER after\_employee\_insert

AFTER INSERT ON employees

FOR EACH ROW

EXECUTE FUNCTION log\_new\_employee();

Test It!

INSERT INTO departments (dept\_name) VALUES ('IT');

INSERT INTO employees (emp\_name, gender, hire\_date, salary, dept\_id, manager\_id)

VALUES ('Alice Smith', 'F', '2024-05-10', 75000, 1, NULL);

SELECT \* FROM audit\_log;

## Transaction

CREATE TABLE staff (

emp\_id SERIAL PRIMARY KEY,

emp\_name VARCHAR(50),

salary NUMERIC

);

INSERT INTO staff (emp\_name, salary)

VALUES ('Alice', 50000),('Bob', 60000),('Charlie', 70000);

## COMMIT

-- Start a transaction

BEGIN;

-- Update Bob’s salary

UPDATE staff SET salary = 65000 WHERE emp\_name = 'Bob';

-- Check the updated data before committing

SELECT \* FROM staff;

-- Make the change permanent

COMMIT;

## ROLLBACK

-- Start another transaction

BEGIN;

-- Try updating Alice’s salary

UPDATE staff SET salary = 100000 WHERE emp\_name = 'Alice';

-- Oops! Realized it was a mistake

ROLLBACK;

SELECT \* FROM staff;

## Savepoint

BEGIN;

-- Do something

SAVEPOINT savepoint\_name;

-- Do more

-- Roll back only to savepoint if needed

ROLLBACK TO SAVEPOINT savepoint\_name;

-- Finalize transaction

COMMIT;

## Example -Savepoint

-- Start the transaction

BEGIN;

-- Step 1: Give Charlie a raise

UPDATE staff SET salary = salary + 10000 WHERE emp\_name = 'Charlie';

-- Set savepoint here

SAVEPOINT after\_charlie\_raise;

-- Step 2: Try to give Bob a huge raise (assume this is a mistake)

UPDATE staff SET salary = salary + 500000 WHERE emp\_name = 'Bob';

-- Realize the mistake and rollback to savepoint

ROLLBACK TO SAVEPOINT after\_charlie\_raise;

-- Now Bob's raise is undone, but Charlie's raise is still there

-- Commit changes so far

COMMIT;