

Project :-

Employee Management Portal

2. Objective

This project aims to create a real-world Employee Management System using PostgreSQL that manages employees, their attendance, salary records, and hierarchical reporting. The system uses advanced SQL features like triggers, functions, constraints, and relational design.

3. Database Structure

The database contains the following main tables:

1. employees – stores personal and job-related info
2. departments – contains department names and locations
3. jobs – defines job titles and roles
4. attendance – tracks employee daily presence
5. salary_history – logs salary changes with triggers
6. leave_requests – manages leave applications

Relationships:

- One department has many employees
- One employee can be a manager of others (self-reference)
- One employee can have many attendance and leave records

4. Key Features

- Self-referencing foreign key for manager system
- Triggers to track salary changes
- Prevent employee from being their own manager

- Function to calculate present days
- Normalized relational design
- Sample data with 15+ entries per table

5. Functions and Triggers Used

1. Trigger: log_salary_changes – saves salary update to history
2. Trigger: prevent_self_manager – prevents assigning self as manager
3. Function: get_present_days(emp_id) – returns total present days

6. Technologies Used

- PostgreSQL
- PL/pgSQL (triggers and functions)
- SQL constraints (foreign key, primary key, not null, unique)

7. Conclusion

The project successfully demonstrates practical use of PostgreSQL for a real-life HR system, including employee hierarchy, attendance management, and salary auditing.

Here are my syntax which i used in my project

Tables

```
create table departments(  
    dept_id serial primary key,  
    dept_name varchar(100)  
);
```

```
create table jobs(  
    job_id serial primary key,
```

```
    job_title text not null,  
    min_salary numeric not null,  
    max_salary numeric not null  
);
```

```
create table employees(  
    emp_id serial primary key,  
    first_name text not null,  
    last_name text not null,  
    email text unique not null,  
    phone text,  
    hire_date date not null,  
    job_id int references jobs(job_id),  
    salary numeric not null,  
    dept_id int references department(dept_id),  
    manager_id int references employees(emp_id)  
);
```

```
create table attendance(  
    att_id serial primary key,  
    emp_id int references employees(emp_id),  
    attendance_date date not null,  
    status text check (status in ('Present', 'Absent', 'Leave')) not null  
);
```

```
create table salary_history(  
    emp_id int references employees(emp_id),  
    salary numeric not null,  
    start_date date not null,  
    end_date date not null,  
    primary key (emp_id, start_date, end_date)
```

record_id serial primary key,

emp_id int references employees(emp_id),

change_date timestamp default now(),

old_salary numeric,

new_salary numeric,

changed_by text

);

Data which i insert in this tables:-

```
SQL Shell (psql)
emp_id | first_name | last_name | email | phone | hire_date | job_id | salary | dept_id | manager_id
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
1 | Ravi | Kumar | ravi.kumar@example.com | 9876500000 | 2020-01-01 | 4 | 120000 | 2 | 
2 | Nidhi | Singh | nidhi.singh@example.com | 9876511111 | 2021-05-01 | 1 | 45000 | 1 | 1
3 | Aman | Joshi | aman.joshi@example.com | 9876522222 | 2022-07-01 | 2 | 60000 | 2 | 1
4 | Preeti | Mehra | preeti.mehra@example.com | 9876533333 | 2022-09-15 | 3 | 50000 | 3 | 1
5 | Rohit | Verma | rohit.verma@example.com | 9876544444 | 2023-01-10 | 2 | 55000 | 2 | 1
6 | Anjali | Yadav | anjali.yadav@example.com | 9876555555 | 2023-02-20 | 1 | 40000 | 1 | 1
7 | Vikram | Shah | vikram.shah@example.com | 9876566666 | 2023-03-12 | 3 | 47000 | 3 | 1
8 | Divya | Kapoor | divya.kapoor@example.com | 9876577777 | 2023-04-08 | 2 | 52000 | 2 | 1
9 | Tarun | Gupta | tarun.gupta@example.com | 9876588888 | 2023-05-15 | 1 | 43000 | 1 | 1
10 | Sana | Sheikh | sana.sheikh@example.com | 9876599999 | 2023-06-01 | 2 | 62000 | 2 | 1
11 | Yash | Chopra | yash.chopra@example.com | 9876510101 | 2023-07-12 | 2 | 67000 | 2 | 1
12 | Priya | Mishra | priya.mishra@example.com | 9876511112 | 2023-08-18 | 1 | 38000 | 1 | 1
(15 rows)

employee_management_db=# select * from departments;
dept_id | dept_name | location
-----+-----+-----
1 | HR | Mumbai
2 | IT | Bangalore
3 | Sales | Delhi
(3 rows)

employee_management_db=# select * from jobs;
job_id | job_title | min_salary | max_salary
-----+-----+-----+-----
1 | HR Executive | 30000 | 60000
2 | Software Engineer | 50000 | 100000
3 | Sales Manager | 40000 | 80000
4 | Team Lead | 80000 | 120000
(4 rows)

employee_management_db=#
```

```
SQL Shell (psql)
1
(15 rows)

employee_management_db=# select * from departments;
 dept_id | dept_name | location
-----+-----+-----
      1 | HR        | Mumbai
      2 | IT        | Bangalore
      3 | Sales     | Delhi
(3 rows)

employee_management_db=# select * from jobs;
 job_id | job_title | min_salary | max_salary
-----+-----+-----+-----
      1 | HR Executive |      30000 |      60000
      2 | Software Engineer |      50000 |     100000
      3 | Sales Manager |      40000 |      80000
      4 | Team Lead   |      80000 |     120000
(4 rows)

employee_management_db=# select * from attendance;
 att_id | emp_id | attendance_date | status
-----+-----+-----+-----
      1 |      2 | 2025-05-01      | Present
      2 |      3 | 2025-05-01      | Present
      3 |      4 | 2025-05-01      | Absent
      4 |      5 | 2025-05-01      | Leave
      5 |      6 | 2025-05-01      | Present
(5 rows)

employee_management_db=# select * from salary_history;
 record_id | emp_id | change_date | old_salary | new_salary | changed_by
-----+-----+-----+-----+-----+-----
(0 rows)

employee_management_db=#
```

Functions which i made :-

-- Trigger Function

CREATE OR REPLACE FUNCTION log_salary_changes()

RETURNS TRIGGER AS \$\$

BEGIN

IF NEW.salary <> OLD.salary THEN

INSERT INTO salary_history (emp_id, old_salary, new_salary, changed_by)

VALUES (OLD.emp_id, OLD.salary, NEW.salary, current_user);

END IF;

RETURN NEW;

END;

\$\$ LANGUAGE plpgsql;

-- Trigger on employees table

```
CREATE TRIGGER trg_salary_change  
AFTER UPDATE ON employees  
FOR EACH ROW  
WHEN (OLD.salary IS DISTINCT FROM NEW.salary)  
EXECUTE FUNCTION log_salary_changes();
```

-- Trigger Function

```
CREATE OR REPLACE FUNCTION prevent_self_manager()  
RETURNS TRIGGER AS $$  
  
BEGIN  
  
    IF NEW.manager_id = NEW.emp_id THEN  
  
        RAISE EXCEPTION 'Employee cannot be their own manager';  
  
    END IF;  
  
    RETURN NEW;  
  
END;  
  
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER trg_prevent_self_manager  
BEFORE INSERT OR UPDATE ON employees  
FOR EACH ROW  
EXECUTE FUNCTION prevent_self_manager();
```

--Function: Calculate total present days of an employee

```
CREATE OR REPLACE FUNCTION get_present_days(p_emp_id INTEGER)  
RETURNS INTEGER AS $$  
  
DECLARE
```

```
total_present INTEGER;

BEGIN

    SELECT COUNT(*) INTO total_present

    FROM attendance

    WHERE emp_id = p_emp_id AND status = 'Present';

    RETURN total_present;

END;

$$ LANGUAGE plpgsql;


SELECT

    e.emp_id,

    e.first_name || ' ' || e.last_name AS employee_name,

    COUNT(a.att_id) FILTER (WHERE a.status = 'Present') AS total_present_days

FROM employees e

LEFT JOIN attendance a ON e.emp_id = a.emp_id

GROUP BY e.emp_id, e.first_name, e.last_name

ORDER BY total_present_days DESC;


CREATE OR REPLACE VIEW employee_attendance_summary AS

SELECT

    e.emp_id,

    e.first_name || ' ' || e.last_name AS employee_name,

    COUNT(a.att_id) FILTER (WHERE a.status = 'Present') AS total_present_days

FROM employees e
```

LEFT JOIN attendance a ON e.emp_id = a.emp_id

GROUP BY e.emp_id, e.first_name, e.last_name;

The screenshot shows the pgAdmin 4 interface with a SQL query executed. The query is as follows:

```
END;  
$$ LANGUAGE plpgsql;  
  
SELECT  
    e.emp_id,  
    e.first_name || ' ' || e.last_name AS employee_name,  
    COUNT(a.att_id) FILTER (WHERE a.status = 'Present') AS total_present_days  
FROM employees e  
LEFT JOIN attendance a ON e.emp_id = a.emp_id  
GROUP BY e.emp_id, e.first_name, e.last_name
```

The results are displayed in a table with the following data:

emp_id [PK] integer	employee_name text	total_present_days bigint
1	6 Anjali Yadav	1
2	3 Aman Joshi	1
3	2 Nidhi Singh	1
4	10 Sana Sheikh	0
5	14 Neha Rastogi	0
6	13 Karan Malhotra	0
7	7 Vikram Shah	0
8	1 Ravi Kumar	0
9	8 Divya Kapoor	0
10	11 Yash Chopra	0
11	9 Tarun Gupta	0
12	15 Abhay Nair	0

Total rows: 15 Query complete 00:00:00.521

The screenshot shows the pgAdmin 4 interface with a SQL query executed. The query is as follows:

```
GROUP BY e.emp_id, e.first_name, e.last_name  
ORDER BY total_present_days DESC;  
  
CREATE OR REPLACE VIEW employee_attendance_summary AS  
SELECT  
    e.emp_id,  
    e.first_name || ' ' || e.last_name AS employee_name,  
    COUNT(a.att_id) FILTER (WHERE a.status = 'Present') AS total_present_days  
FROM employees e  
LEFT JOIN attendance a ON e.emp_id = a.emp_id  
GROUP BY e.emp_id, e.first_name, e.last_name;  
  
SELECT * FROM employee_attendance_summary;
```

The results are displayed in a table with the following data:

emp_id integer	employee_name text	total_present_days bigint
1	15 Abhay Nair	0
2	5 Rohit Verma	0
3	4 Preeti Mehra	0
4	10 Sana Sheikh	0
5	6 Anjali Yadav	1
6	14 Neha Rastogi	0
7	13 Karan Malhotra	0
8	2 Nidhi Singh	1
9	7 Vikram Shah	0

Total rows: 15 Query complete 00:00:00.111

Thank you