Practice Dec17

Table Structures

1. **Employees**
   - Represents employees in the company.
   - Employees belong to one department.

   Column Name Data Type Description `employee_id` INT (PK) Unique identifier for each employee `first_name` VARCHAR(50) Employee's first name `last_name` VARCHAR(50) Employee's last name `department_id` INT (FK) Department to which the employee belongs `hire_date` DATE Date of hire `salary` DECIMAL(10,2) Employee's salary
2. **Departments**
   - Lists all departments in the company.

   Column Name Data Type Description `department_id` INT (PK) Unique identifier for each department `department_name` VARCHAR(100) Name of the department `manager_id` INT (FK) Manager (employee) for the department
3. **Projects**
   - Represents projects employees work on.

   Column Name Data Type Description `project_id` INT (PK) Unique identifier for each project `project_name` VARCHAR(100) Name of the project `start_date` DATE Start date of the project `end_date` DATE End date of the project
4. **Employee_Projects**
   - Represents assignments of employees to projects (many-to-many relationship).

   Column Name Data Type Description `employee_id` INT (FK) Reference to the employee `project_id` INT (FK) Reference to the project `hours_logged` DECIMAL(6,2) Total hours logged for the project `allocation_date` DATE Date when the allocation started
5. **Transactions**
   - Represents payments or transactions related to employee salaries.

Column Name Data Type Description `transaction_id` INT (PK) Unique identifier for each transaction `employee_id` INT (FK) Employee for whom the transaction applies `amount` DECIMAL(10,2) Transaction amount `transaction_date` DATE Date of the transaction `transaction_type` VARCHAR(50) Type: Salary, Bonus, Deduction

---

Practice Questions

Set 1: Self Joins, Aggregations, and Conditional Queries

1. Write a query to fetch the names of employees and their department name.
2. Find the total hours logged by each employee across all projects.
3. Retrieve the department name and count of employees in each department.
4. Identify employees who work on more than 2 projects.
5. Write a query to find employees who do not work on any project.
6. Find employees earning a salary greater than the average salary of their department.
7. Write a query using a **self join** to display employee names along with their manager's name.
8. Identify projects where the total hours logged by all employees exceed 500.
9. Retrieve the name of the department where the manager has the highest salary.
10. Display employees who have logged more than 40 hours on any project using a CASE statement to classify them as 'Overtime' or 'Normal.'
11. Find employees who were hired after January 1, 2020, and have not been allocated to any projects.
12. Write a query to display employees whose total salary transactions (including bonuses/deductions) exceed 10,000.
13. Retrieve the transaction details of employees where the transaction type is "Bonus" and the amount is in the top 5% of all bonuses.
14. List employees whose salary is the highest in their respective department.
15. Display the average hours logged by employees for each project.
16. Write a query to fetch all employees who belong to the same department as "John Doe" using self joins.
17. Find the departments where no projects have started.
18. Retrieve employee details for employees working on all projects.

19. List employees whose names start with the letter 'A' and have logged more than 20 hours on any project.
20. Identify the total transactions per employee along with their latest transaction date.

Set 2: Advanced Queries with Joins, CASE, Aggregation

1. Write a query to list employees, their projects, and hours logged, with projects having more than 100 hours total.
2. Find employees with the most logged hours across all projects.
3. Display department names and the total salary paid to employees in each department.
4. Retrieve employees whose salaries are above the average salary across the company.
5. Write a query to display projects that started after 2023 and do not have any employees assigned yet.
6. Using a **CASE statement**, classify transactions as "High Value" if the amount is greater than 5,000; otherwise, classify them as "Low Value."
7. Identify the project(s) with the most employees assigned.
8. Display departments and the count of employees earning below 50,000.
9. Write a query to find the 5 employees with the **highest salaries** who have worked on the most projects.
10. Retrieve project names and the average hours logged per employee for each project.
11. Write a query to display transaction amounts for employees hired in 2022 or later.
12. Identify employees who have not logged any hours on projects but have received salary transactions.
13. Find departments with no employees using a **LEFT JOIN**.
14. Retrieve the employees who have worked on more than one project and logged over 200 hours in total.
15. Write a query to fetch projects where no hours have been logged by employees.
16. Display employees who have received both a **bonus** and a **deduction** in their transactions.
17. Identify employees with missing or null `last_name` values and classify them as "Data Issue."

18. Using a **self join**, display pairs of employees in the same department with different salaries.
19. Write a query to calculate the total hours logged for each employee and classify them into "High Performer" (> 100 hours) or "Normal Performer."
20. Find the latest transaction date for each employee who received a salary payment.

Test Data Requirements (Use any AI tool for this)

1. Create the table as mentioned in Table Structures section
2. Populate the tables with a minimum of **100 rows** in the `Employees` table, ensuring diversity in departments, salaries, and hire dates.
3. Insert at least **50 rows** into the `Departments` table with varied department names and managers.
4. Add **50-70 rows** in the `Projects` table with projects spanning different dates and durations.
5. Populate the `Employee_Projects` table to create realistic many-to-many mappings, ensuring some employees work on multiple projects and others on none.
6. Insert **200 rows** into the `Transactions` table with salary, bonus, and deduction transactions spread across employees.