MySQL Assignment: Stored Procedure, Triggers

Objective:

This assignment will help you practice working with **Stored Procedures, Triggers, and Views** using MySQL.

Part 1: Stored Procedures

Task 1: Insert New Employee

Create a **stored procedure** `sp_add_employee` that inserts a new employee into the `Employees` table. The procedure should accept **first name, last name, email, hire date, salary, and department ID** as parameters.

Example Call:

```
CALL sp_add_employee('Raj', 'Kumar', 'raj.kumar@example.com', '2023-08-10', 70000, 1);
```

Task 2: Update Employee Salary

Create a **stored procedure** `sp_update_salary` that increases an employee's salary by a given percentage. The procedure should accept **employee ID and percentage increment** as parameters.

Example Call:

```
CALL sp_update_salary(1, 10);
```

(Will increase **Amit Sharma's** salary by 10%)

Task 3: Delete Employee and Log Action

Create a **stored procedure** `sp_delete_employee` that removes an employee from the `Employees` table and logs the deleted employee's details into a separate table called `DeletedEmployees`.

1. First, create the `DeletedEmployees` table:

   ```
   CREATE TABLE DeletedEmployees (
   ```

```
    emp_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(100),
    hire_date DATE,
    salary DECIMAL(10,2),
    dept_id INT,
    deleted_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

2. Then, create the procedure. The procedure should:
    - Insert the employee details into `DeletedEmployees` before deleting them from `Employees`.

Example Call:

```
CALL sp_delete_employee(3);
```

Part 2: Triggers

Task 4: Salary Constraint Trigger

Create a **BEFORE INSERT** trigger `before_salary_check` that ensures no employee is inserted with a salary less than 30,000. If an attempt is made, it should generate an error message.

Task 5: Log Salary Changes

Create an **AFTER UPDATE** trigger `after_salary_update` that records changes in salary into a table `SalaryHistory`.

1. First, create the `SalaryHistory` table:

```
CREATE TABLE SalaryHistory (
    emp_id INT,
    old_salary DECIMAL(10,2),
    new_salary DECIMAL(10,2),
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

2. Then, create a trigger that logs the previous and updated salary whenever an employee's salary is changed.

Part 3: Views

Task 6: View for Highest Salary Employees

Create a **View** `high_salary_employees` to display **employees earning above 75,000**, along with their department name.

Expected Output Example:

emp_id	first_name	last_name	salary	dept_name
3	Rohit	Patel	80000	Finance
4	Sneha	Iyer	90000	IT

Task 7: View for Salespersons Handling Most Orders

(Not applicable since there's no `Salesperson` or `Orders` table, you can skip this or modify it to count employees in each department.)

Task 8: View for Employees Without Department

Create a **View** `employees_without_dept` to display employees who do not belong to any department.

Expected Output Example:

emp_id	first_name	last_name	salary
5	Vikram	Singh	55000

Submission Guidelines:

1. Save all **stored procedures, triggers, and views** in a `.sql` file.
2. Ensure queries are **well-structured and formatted**.
3. Test all procedures and triggers before submission.