ASSIGNMENT-2

BY: P.Venkata Sai Pradeep Reddy 192311364

QUESTION :

Create a database to manage employees, departments, payroll, and performance evaluations.

- Model tables for employees, departments, payroll, and performance evaluations.

- Write stored procedures for updating payroll and handling performance reviews.

- Implement triggers to update department budgets when payroll changes.

- Write SQL queries to generate employee performance reports.

To manage employees, departments, payroll, and performance evaluations, you can design the following SQL schema with necessary tables, stored procedures, triggers, and queries.

Step 1: Database Schema

1. Departments Table:

This table stores information about the departments.

```
CREATE TABLE departments (

   department_id INT PRIMARY KEY AUTO_INCREMENT,

   department_name VARCHAR(100) NOT NULL,

   department_budget DECIMAL(15, 2) NOT NULL

);
```

2. Employees Table:

This table stores information about employees.

```
CREATE TABLE employees (

   employee_id INT PRIMARY KEY AUTO_INCREMENT,

   first_name VARCHAR(50) NOT NULL,
   last_name VARCHAR(50) NOT NULL,
   department_id INT,
   salary DECIMAL(15, 2),
   hire_date DATE,
   performance_rating INT,
   FOREIGN KEY (department_id) REFERENCES departments(department_id)
);
```

3. Payroll Table:

This table stores payroll information for employees.

```
CREATE TABLE payroll (
    payroll_id INT PRIMARY KEY AUTO_INCREMENT,
    employee_id INT,
    pay_date DATE,
    salary DECIMAL(15, 2),
    FOREIGN KEY (employee_id) REFERENCES employees(employee_id)
);
```

4. Performance Evaluations Table:

This table stores employee performance reviews.

```
CREATE TABLE performance_evaluations (
    evaluation_id INT PRIMARY KEY AUTO_INCREMENT,
    employee_id INT,
    evaluation_date DATE,
    score INT,
    feedback TEXT,

    FOREIGN KEY (employee_id) REFERENCES employees(employee_id)

);
```

Step 2: Stored Procedures

1. Updating Payroll:

This stored procedure will update the salary in the payroll and employees table when there is a change in the payroll.

```
DELIMITER $$


CREATE PROCEDURE update_payroll(

    IN emp_id INT,

    IN new_salary DECIMAL(15, 2)

)

BEGIN

    -- Update employee salary

    UPDATE employees

    SET salary = new_salary

    WHERE employee_id = emp_id;


    -- Insert into payroll table

    INSERT INTO payroll (employee_id, pay_date, salary)

    VALUES (emp_id, CURDATE(), new_salary);

END $$


DELIMITER ;
```

2. Handling Performance Reviews:

This stored procedure inserts a performance review for an employee.

```
DELIMITER $$


CREATE PROCEDURE add_performance_review(
    IN emp_id INT,
    IN score INT,
    IN feedback TEXT
)
BEGIN
    INSERT INTO performance_evaluations (employee_id, evaluation_date, score, feedback)
    VALUES (emp_id, CURDATE(), score, feedback);


    -- Update performance rating of the employee
    UPDATE employees
    SET performance_rating = score
    WHERE employee_id = emp_id;
END $$


DELIMITER ;
```

Step 3: Triggers

1. Update Department Budget when Payroll Changes:

This trigger ensures that the department's budget is adjusted whenever an employee's salary is updated.

```
DELIMITER $$


CREATE TRIGGER update_department_budget
AFTER UPDATE ON employees
FOR EACH ROW
BEGIN
    DECLARE total_payroll DECIMAL(15, 2);


    -- Calculate total payroll for the department
    SELECT SUM(salary) INTO total_payroll
    FROM employees
    WHERE department_id = OLD.department_id;


    -- Update department budget
    UPDATE departments
    SET department_budget = total_payroll
    WHERE department_id = OLD.department_id;
END $$
DELIMITER ;
```

Step 4: SQL Queries

1. Generate Employee Performance Reports:

This query generates a report of employee performance reviews, including the employee details and their most recent performance review.

```
SELECT e.employee_id, e.first_name, e.last_name, e.department_id, e.salary, e.performance_rating,
p.evaluation_date, p.score, p.feedback
```

```
FROM employees e
JOIN performance_evaluations p ON e.employee_id = p.employee_id
WHERE p.evaluation_date = (
   SELECT MAX(evaluation_date)
   FROM performance_evaluations
   WHERE employee_id = e.employee_id
)
ORDER BY e.department_id, e.last_name;
```

2. Generate Payroll Summary:

This query generates a payroll summary for all employees.

```
SELECT e.employee_id, e.first_name, e.last_name, e.department_id, p.pay_date, p.salary
FROM employees e
JOIN payroll p ON e.employee_id = p.employee_id
ORDER BY p.pay_date DESC;
```

3. Generate Department Budget Report:

This query shows the total payroll of each department along with the current department budget.

```
SELECT d.department_name, d.department_budget, SUM(e.salary) AS total_payroll
FROM departments d
JOIN employees e ON d.department_id = e.department_id
GROUP BY d.department_id;
```

Example Workflow:

When an employee gets a salary increase, you call the stored procedure update_payroll to update the salary and record the change in the payroll table.

If an employee undergoes a performance evaluation, you call the add_performance_review stored procedure to record the review and update the performance rating.

Whenever an employee's salary changes, the trigger update_department_budget updates the department's total payroll and adjusts its budget accordingly.

You can run the SQL queries above to generate reports on employee performance, payroll details, or department budgets.

This setup allows for a comprehensive, dynamic, and responsive employee management system.

To illustrate the scenario, let's work with a set of example data and create the tables for employees, departments, payroll, and performance evaluations.

1. Departments Table:

We'll create a few departments.

```
CREATE TABLE departments (
    department_id INT PRIMARY KEY AUTO_INCREMENT,
    department_name VARCHAR(100) NOT NULL,
    department_budget DECIMAL(15, 2) NOT NULL
);

-- Example data for departments
INSERT INTO departments (department_name, department_budget) VALUES
('HR', 500000.00),
('IT', 1000000.00),
('Sales', 750000.00);
```

2. Employees Table:

Now, we’ll create employees and assign them to departments. They will have some initial salary and performance rating.

```
CREATE TABLE employees (
    employee_id INT PRIMARY KEY AUTO_INCREMENT,
    first_name VARCHAR(50) NOT NULL,

    last_name VARCHAR(50) NOT NULL,

    department_id INT,

    salary DECIMAL(15, 2),

    hire_date DATE,

    performance_rating INT,

    FOREIGN KEY (department_id) REFERENCES departments(department_id)

);


-- Example data for employees

INSERT INTO employees (first_name, last_name, department_id, salary, hire_date, performance_rating)
VALUES

('John', 'Doe', 1, 55000.00, '2020-01-15', 4),

('Jane', 'Smith', 2, 80000.00, '2019-05-22', 5),

('Alice', 'Brown', 3, 60000.00, '2021-09-10', 3);
```

3. Payroll Table:

Let’s create the payroll table and populate it with initial payroll records.

```
CREATE TABLE payroll (

    payroll_id INT PRIMARY KEY AUTO_INCREMENT,

    employee_id INT,

    pay_date DATE,

    salary DECIMAL(15, 2),

    FOREIGN KEY (employee_id) REFERENCES employees(employee_id)

);
```

```
-- Example data for payroll

INSERT INTO payroll (employee_id, pay_date, salary) VALUES

(1, '2023-10-31', 55000.00),

(2, '2023-10-31', 80000.00),

(3, '2023-10-31', 60000.00);
```

4. Performance Evaluations Table:

We'll add performance evaluations for employees, including scores and feedback.

```
CREATE TABLE performance_evaluations (

    evaluation_id INT PRIMARY KEY AUTO_INCREMENT,

    employee_id INT,

    evaluation_date DATE,

    score INT,

    feedback TEXT,

    FOREIGN KEY (employee_id) REFERENCES employees(employee_id)

);


-- Example data for performance evaluations

INSERT INTO performance_evaluations (employee_id, evaluation_date, score, feedback) VALUES

(1, '2023-12-01', 4, 'Great teamwork and communication skills.'),

(2, '2023-12-01', 5, 'Outstanding performance and leadership.'),

(3, '2023-12-01', 3, 'Needs improvement in project management skills.');
```

Step 5: Running Stored Procedures with Example Data

1. Updating Payroll:

For example, if John Doe gets a salary increase, you would call the update_payroll stored procedure like this:

```sql
CALL update_payroll(1, 60000.00);  -- Increase John's salary to 60000
```

This will update the salary in the employees table and insert a record into the payroll table for John Doe.

2. Adding a Performance Review:

For example, if Jane Smith receives a new performance review, you would call the add_performance_review stored procedure:

```sql
CALL add_performance_review(2, 5, 'Excellent leadership and problem-solving skills.');
```

This will insert a new performance evaluation record for Jane and update her performance rating.

Step 6: Triggers Example

1. Trigger to Update Department Budget:

Let’s say you increase Alice Brown’s salary to $70,000, and we want the department budget for Sales (department_id = 3) to be updated automatically:

```sql
UPDATE employees SET salary = 70000 WHERE employee_id = 3;
```

The trigger update_department_budget will automatically update the Sales department budget by recalculating the total payroll for the department.

Step 7: Queries with Example Data

1. Generate Employee Performance Reports:

This query will list employees' performance evaluations, focusing on their most recent review.

```sql
SELECT e.employee_id, e.first_name, e.last_name, e.department_id, e.salary, e.performance_rating, p.evaluation_date, p.score, p.feedback

FROM employees e
```

```
JOIN performance_evaluations p ON e.employee_id = p.employee_id

WHERE p.evaluation_date = (

   SELECT MAX(evaluation_date)

   FROM performance_evaluations

   WHERE employee_id = e.employee_id

)

ORDER BY e.department_id, e.last_name;
```

2. Generate Payroll Summary:

This query will show payroll details for all employees.

```
SELECT e.employee_id, e.first_name, e.last_name, e.department_id, p.pay_date, p.salary

FROM employees e

JOIN payroll p ON e.employee_id = p.employee_id

ORDER BY p.pay_date DESC;
```

3. Generate Department Budget Report:

This query shows the total payroll for each department and compares it with the department’s current budget.

```
SELECT d.department_name, d.department_budget, SUM(e.salary) AS total_payroll

FROM departments d

JOIN employees e ON d.department_id = e.department_id

GROUP BY d.department_id;
```

Summary of Example Data:

Departments: HR, IT, and Sales.

Employees: John Doe, Jane Smith, Alice Brown.

Payroll: Payroll records for the employees with their salaries.

Performance Evaluations: Recent performance evaluations with scores and feedback.

By running these queries and stored procedures, you can manage employee payroll, performance evaluations, department budgets, and generate reports for analysis

Logical, Physical, and Conceptual ER Diagrams

To illustrate the relationships between the entities (tables) and their attributes, we'll provide the following three types of ER diagrams:

Conceptual ER Diagram (High-level overview)

Logical ER Diagram (With primary keys, foreign keys, and relationships)

Physical ER Diagram (Showing specific details about the data types and the table structure)

1. Conceptual ER Diagram

The Conceptual ER Diagram provides a high-level overview of the entities (tables) and their relationships without focusing on implementation details like primary/foreign keys.

Entities:

Department: Manages employee data for each department.

Employee: Stores employee details.

Payroll: Represents payroll records for employees.

Performance Evaluation: Stores performance reviews for employees.

Relationships:

Department "has" many Employees.

Employee "has" many Payroll Records.

Employee "receives" many Performance Evaluations.

```
+-----------------+       +----------------+

|  Department    |       |    Employee    |
+-----------------+       +----------------+
| department_id PK|       | employee_id PK |
| department_name |       | first_name     |
| department_budget|      | last_name      |
+-----------------+       | salary         |
     |              | hire_date      |
     |              | performance_rating |
     |              | department_id FK |
     |              +----------------+
     |                  |
     |                  |
     |                  |
+----------------+        +------------------------+
|    Payroll    |         | Performance Evaluation |
+----------------+        +------------------------+
| payroll_id PK  |        | evaluation_id PK       |
| employee_id FK |        | employee_id FK         |
| pay_date       |        | evaluation_date        |
| salary         |        | score                  |
+----------------+        | feedback               |
                  +------------------------+
```

2. Logical ER Diagram

The Logical ER Diagram refines the conceptual diagram by specifying primary keys (PK), foreign keys (FK), and the relationships between entities. It also outlines how each entity (table) relates to one another.

Entities:

Department: department_id, department_name, department_budget

Employee: employee_id, first_name, last_name, salary, hire_date, performance_rating, department_id

Payroll: payroll_id, employee_id, pay_date, salary

Performance Evaluation: evaluation_id, employee_id, evaluation_date, score, feedback

```
+-----------------+      +----------------+
|  Department   |      |   Employee   |
+-----------------+      +----------------+
| department_id PK|◄───────┐ | employee_id PK |
| department_name |      | | first_name    |
| department_budget|     | | last_name     |
+-----------------+     | | salary        |
    ▲             | | hire_date     |
    |             | | performance_rating |
    |             | | department_id FK (Dept)|
    |             | +----------------+
    |             |
+----------------+      |
|    Payroll    |      |
+----------------+      |
| payroll_id PK  |      |
| employee_id FK |───────┘
| pay_date      |
| salary        |
+----------------+
      |

      |

+------------------------+

| Performance Evaluation |

+------------------------+

| evaluation_id PK       |

| employee_id FK         |

| evaluation_date        |

| score                  |

| feedback               |

+------------------------+
```

Relationships:

One Department has many Employees (1-to-many relationship).

One Employee has many Payroll records (1-to-many relationship).

One Employee has many Performance Evaluations (1-to-many relationship).

3. Physical ER Diagram

The Physical ER Diagram gives a more detailed view of how the data will be stored in the database. It includes specific data types for each column and outlines primary and foreign key constraints.

```
+------------------------------+

|      Department              |

+------------------------------+

| department_id INT PK         |

| department_name VARCHAR(100) |

| department_budget DECIMAL(15, 2)|

+------------------------------+
       ▲
       |
+------------------------------+
|        Employee              |
+------------------------------+
| employee_id INT PK           |
| first_name VARCHAR(50)       |
| last_name VARCHAR(50)        |
| salary DECIMAL(15, 2)        |
| hire_date DATE               |
| performance_rating INT       |
| department_id INT FK         |
+------------------------------+
       |
+------------------------------+
|          Payroll             |
+------------------------------+
| payroll_id INT PK            |
| employee_id INT FK           |
| pay_date DATE                |
| salary DECIMAL(15, 2)        |
+------------------------------+
       |
+------------------------------+
| Performance Evaluation       |
+------------------------------+
| evaluation_id INT PK         |
| employee_id INT FK           |
| evaluation_date DATE         |
| score INT                    |
| feedback TEXT                |
+------------------------------+
```

Explanation of Data Types:

INT: Integer (e.g., employee_id, department_id, evaluation_id).

VARCHAR: Variable-length string (e.g., department_name, first_name, last_name).

DECIMAL(15,2): Decimal numbers with up to 15 digits and 2 decimal places (e.g., salary, department_budget).

DATE: Date type (e.g., hire_date, pay_date, evaluation_date).

TEXT: Long text (e.g., feedback).

Key Points:

The conceptual diagram shows the broad relationships between entities.

The logical diagram adds the keys and specific relationships (foreign key links).

The physical diagram is the implementation level where we define actual column types and constraints to create the tables in the database.

CONCLISION:

This approach will help you visualize the structure of your system at various levels of abstraction, ensuring a solid understanding of both the relationships and technical details required for building the database.