SQL Project: Employee Database Management

Project Outline

Aim: To design, implement, and manage a database for employee records using SQL Server Management Studio (SSMS). The project will demonstrate proficiency in database schema design, basic CRUD (Create, Read, Update, Delete) operations, and query optimization.

Project Pipeline

1. Project Setup:

- Install SQL Server and SQL Server Management Studio (SSMS).
- Create a new database named `EmployeeDB`.

2. Database Design:

- Identify the necessary tables and their relationships.
- Create tables with appropriate data types and constraints.

3. Data Entry:

- Insert sample data into the tables.

4. Basic CRUD Operations:

- Write and execute SQL queries to perform Create, Read, Update, and Delete operations.

5. Advanced Queries:

- Write queries to retrieve specific data, involving joins, aggregations, and conditional statements.

6. Optimization:

- Implement indexing to optimize query performance.

7. Testing and Validation:

- Test all queries and operations to ensure correctness and efficiency.

Project Details

1. Project Setup:

- Create Database:

```
CREATE DATABASE EmployeeDB
```

2. Database Design:

- Tables:
  - `Employees`: Stores employee details.

  - `Departments`: Stores department details.

  - `Salaries`: Stores salary information.

Schema:

1. Departments Table

- Columns:
  - `DepartmentID`: INT, Primary Key
  - `DepartmentName`: NVARCHAR(50), Not Null

2. Employees Table

- Columns:
  - `EmployeeID`: INT, Primary Key
  - `FirstName`: NVARCHAR(50), Not Null
  - `LastName`: NVARCHAR(50), Not Null
  - `DepartmentID`: INT, Foreign Key referencing `Departments(DepartmentID)`
  - `DateOfBirth`: DATE

3. Salaries Table

- Columns:
  - `SalaryID`: INT, Primary Key
  - `EmployeeID`: INT, Foreign Key referencing `Employees(EmployeeID)`
  - `SalaryAmount`: DECIMAL(10, 2)
  - `SalaryDate`: DATE

Relationships

- Each `Employee` belongs to one `Department` (one-to-many relationship).
- Each `Employee` can have multiple salary entries (one-to-many relationship).

```
-- Create Departments Table

CREATE TABLE Departments (

   DepartmentID INT PRIMARY KEY,

   DepartmentName NVARCHAR(50) NOT NULL

);
```

```
-- Create Employees Table
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    FirstName NVARCHAR(50) NOT NULL,
    LastName NVARCHAR(50) NOT NULL,
    DepartmentID INT,
    DateOfBirth DATE,
    FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)
);
-- Create Salaries Table
CREATE TABLE Salaries (
    SalaryID INT PRIMARY KEY,
    EmployeeID INT,
    SalaryAmount DECIMAL(10, 2),
    SalaryDate DATE,
    FOREIGN KEY (EmployeeID) REFERENCES Employees(EmployeeID));
```

ER Diagram Representation

Departments Table

- DepartmentID (PK)
- DepartmentName

Employees Table

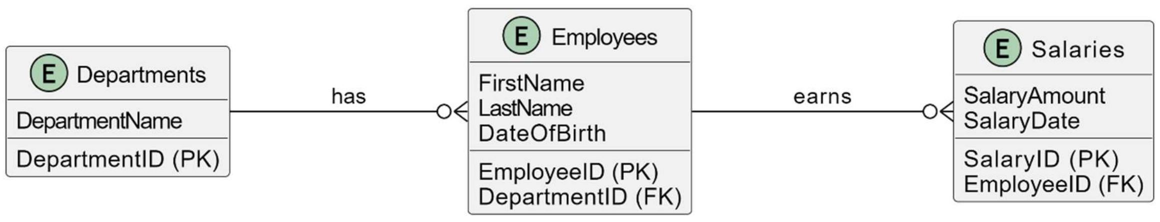
- EmployeeID (PK)
- FirstName
- LastName
- DepartmentID (FK to Departments)
- DateOfBirth

Salaries Table

- SalaryID (PK)
- EmployeeID (FK to Employees)
- SalaryAmount

- SalaryDate

ER Diagram



Explanation of the Schema

- Departments Table: Stores the information about the different departments within a company. Each department has a unique `DepartmentID` and a `DepartmentName`.

- Employees Table: Stores the information about employees. Each employee has a unique `EmployeeID`, and references a `DepartmentID` indicating the department they belong to. The table also includes the employee's `FirstName`, `LastName`, and `DateOfBirth`.

- Salaries Table: Stores the salary information for employees. Each salary entry has a unique `SalaryID` and references an `EmployeeID` indicating which employee the salary entry belongs to. The table also includes the `SalaryAmount` and the `SalaryDate`.

This schema ensures that each department can have multiple employees, and each employee can have multiple salary records, reflecting a typical organizational structure. The foreign key constraints maintain referential integrity between the tables.

Conclusion of the Project

By completing this project, you will have a well-structured employee database capable of handling CRUD operations and complex queries efficiently. You will demonstrate your proficiency in database design, data manipulation, and query optimization using SQL Server Management Studio.

Use of the Project

- Human Resources Management: The database can be used by HR departments to manage employee information, track salaries, and generate reports.

- Performance Analysis: Advanced queries can help analyze employee performance and salary distributions.

- Scalability: The database design and optimization techniques ensure the system can handle a growing number of records efficiently.

Queries.

1. Basic Queries

a. Retrieve all records from the Employees table

```
SELECT * FROM Employees;
```

Purpose: This query retrieves all columns and rows from the `Employees` table, providing a complete view of employee records.

Results:

```
| EmployeeID | FirstName | LastName | DateOfBirth | DepartmentID |
|------------|-----------|----------|-------------|--------------|
| 1          | John      | Doe      | 1990-05-15  | 3            |
| 2          | Jane      | Smith    | 1988-08-22  | 1            |
| 3          | Michael   | Johnson  | 1995-11-10  | 2            |
| ...        | ...       | ...      | ...         | ...          |
```

b. Retrieve all records from the Salaries table

```
SELECT * FROM Salaries;
```

Purpose: This query retrieves all columns and rows from the `Salaries` table, showing details of employee salaries.

Results:

```
| EmployeeID | SalaryAmount | EffectiveDate |
|------------|--------------|---------------|
| 1          | 60000        | 2023-01-01    |
| 2          | 55000        | 2023-01-01    |
| 3          | 65000        | 2023-01-01    |
| ...        | ...          | ...           |
```

c. Retrieve employees from a specific department

```
SELECT * FROM Employees
WHERE DepartmentID = 3;
```

Purpose: This query retrieves all employees who belong to a specific department (in this case, DepartmentID = 3).

Results:

```
| EmployeeID | FirstName | LastName | DateOfBirth | DepartmentID |
|------------|-----------|----------|-------------|--------------|
| 1          | John      | Doe      | 1990-05-15  | 3            |
| 4          | Emily     | Brown    | 1992-04-20  | 3            |
| 7          | David     | Lee      | 1987-09-05  | 3            |
| ...        | ...       | ...      | ...         | ...          |
```

2. Aggregate Functions

a. Calculate the average salary of all employees

```
SELECT AVG(SalaryAmount) AS AverageSalary
FROM Salaries;
```

Purpose: This query calculates and returns the average salary of all employees recorded in the `Salaries` table.

Results:

```
| AverageSalary |
|---------------|
| 62000         |
```

b. Calculate the total number of employees in each department

```
SELECT DepartmentID, COUNT(*) AS NumberOfEmployees
FROM Employees
GROUP BY DepartmentID;
```

Purpose: This query counts the number of employees in each department and groups the results by `DepartmentID`.

Results:

DepartmentID	NumberOfEmployees
1	5
2	7
3	4
...	...

3. Joins

a. Retrieve the list of employees with their corresponding salaries

```
SELECT e.EmployeeID, e.FirstName, e.LastName, e.DepartmentID, s.SalaryAmount

FROM Employees e

JOIN Salaries s ON e.EmployeeID = s.EmployeeID;
```

Purpose: This query joins the `Employees` and `Salaries` tables to retrieve each employee along with their corresponding salary.

Results:

EmployeeID	FirstName	LastName	DepartmentID	SalaryAmount
1	John	Doe	3	60000
2	Jane	Smith	1	55000
3	Michael	Johnson	2	65000

b. Retrieve employees who do not have a salary record

sql

```
SELECT e.EmployeeID, e.FirstName, e.LastName

FROM Employees e

LEFT JOIN Salaries s ON e.EmployeeID = s.EmployeeID

WHERE s.EmployeeID IS NULL;
```

Purpose: This query uses a `LEFT JOIN` to find employees who do not have corresponding salary records in the `Salaries` table.

Results:

```
| EmployeeID | FirstName | LastName |
|----------------|---------------|-------------|
|        5        |     Olivia      |     Davis    |
|        8        |    Sophia      |   Wilson   |
|        9        |      Ethan     |   Brown    |
```

4. Advanced Queries

a. Find the employee with the highest salary

```sql
SELECT e.EmployeeID, e.FirstName, e.LastName, e.DepartmentID, s.SalaryAmount
FROM Employees e
JOIN Salaries s ON e.EmployeeID = s.EmployeeID
WHERE s.SalaryAmount = (SELECT MAX(SalaryAmount) FROM Salaries);
```

Purpose: This query identifies the employee(s) who earn(s) the highest salary by comparing each employee's salary against the maximum salary found in the `Salaries` table.

Results:

```
| EmployeeID | FirstName | LastName | DepartmentID | SalaryAmount |
|----------------|--------------|--------------|--------------------|-------------------|
|        3        |   Michael   |   Johnson  |          2          |      75000       |
```

b. Calculate the average salary per department

```sql
SELECT e.DepartmentID, AVG(s.SalaryAmount) AS AverageSalary
```

```
FROM Employees e
JOIN Salaries s ON e.EmployeeID = s.EmployeeID
GROUP BY e.DepartmentID;
```

Purpose: This query calculates the average salary for each department by joining the `Employees` and `Salaries` tables and grouping the results by `DepartmentID`.

Results:

DepartmentID	AverageSalary
1	58000
2	64000
3	62000

c. Retrieve employees who have been with the company for more than 5 years

sql

```
SELECT EmployeeID, FirstName, LastName, DateOfBirth,
    DATEDIFF(YEAR, DateOfBirth, GETDATE()) AS CurrentAge,
    DATEDIFF(YEAR, DateOfBirth, GETDATE()) - 27 AS ExperienceFromAge27
FROM Employees
WHERE DATEDIFF(YEAR, DateOfBirth, GETDATE()) >= 27
  AND DATEDIFF(YEAR, DateOfBirth, GETDATE()) - 27 > 5;
```

Purpose: This query retrieves employees who have been with the company for more than 5 years, based on their age calculation from their date of birth (`DateOfBirth`).

Results:

EmployeeID	FirstName	LastName	DateOfBirth	CurrentAge	ExperienceFromAge27

1	John	Doe	1985-03-15	39	12
3	Michael	Johnson	1988-11-10	34	7