**UNIVERSITY INSTITUTE OF COMPUTING**

**CASE STUDY REPORT**

**ON**

**PARTICULAR CASE STUDY**

Program Name: BCA

Subject Name/Code: Database Management System (23CAT-251)

**Submitted by: Submitted to:**

**Name:** Priksha Thakur **Name: M/s. Sonia**

**UID:** 23Bca10483 **Designation: Professor**

**Section:** 8

’B

ABSTRACT

* Introduction:

The **Salary Management System (SMS)** is a centralized, database-driven application designed to streamline the administration of employee-related operations in an organization. As businesses grow and the number of employees increases, managing their records, salary structures, and departmental affiliations becomes complex and prone to errors if done manually. This system solves these problems by providing an efficient, digital solution that handles employee data, salary calculations, departmental assignments, and position tracking with accuracy and speed.

The SMS database focuses on four major components:

**Employee Management** – Storing and managing employee personal details, employment dates, gender, and contact information.

**Departmental Relationships** – Associating employees with specific departments based on their roles and responsibilities.

**Position Tracking** – Assigning and managing the job titles or roles held by employees within the company.

**Salary Computation** – Automatically calculating employee salaries based on their basic pay, bonus, and deductions using SQL’s computed column capabilities.

The system is designed not only to allow for the **addition, update, and deletion** of records, but also to **generate reports** for analysis such as department-wise salary totals, top earners, and salary averages. It uses SQL for querying and reporting, offering insights into the organization’s payroll structure. The SMS is scalable and can be integrated with broader Human Resource systems or financial applications, supporting long-term business operations.

By automating the storage and retrieval of employee-related data, this system eliminates redundancy, maintains consistency, and ensures that all stakeholders—from HR executives to accountants—can access timely and accurate information.

* Technique:

The **Salary Management System (SMS)** is developed using **MySQL**, applying the principles of relational database management and normalization to reduce redundancy and maintain data consistency. The database is designed using multiple interrelated tables such as Employee, Department, Salary, Position, and EmployeeDepartment. Each table is linked via **foreign keys** to enforce referential integrity, ensuring that all data remains synchronized across the system.

The following key techniques have been implemented in the SMS:

### 1. ****Data Insertion and Structuring****

Populated with sample data for employees, salaries, departments, and positions.

Uses CHECK constraints (e.g., gender restriction) and UNIQUE constraints (e.g., email uniqueness) for data validation.

Ensures referential integrity using FOREIGN KEY constraints with ON DELETE CASCADE behavior.

### 2. ****Relational Design & Normalization****

The schema is normalized to at least **Third Normal Form (3NF)**, ensuring each piece of data is stored only once.

This minimizes duplication and supports efficient data retrieval.

### 3. ****Querying and Data Retrieval****

Various SELECT queries retrieve specific or aggregate information.

**JOIN operations** combine data across multiple tables to show real-time relational insights (e.g., employee + salary + department).

**Aggregation functions** such as SUM, AVG, and COUNT are used for financial reports.

### 4. ****Updating and Deleting Records****

Supports updates to employee or salary details using UPDATE queries.

Cascading delete ensures that deleting an employee also removes their salary, position, and department records automatically.

### 5. ****Computed Fields for Automation****

The total\_salary field in the Salary table is **automatically calculated** using the expression:  
basic\_salary + bonus - deductions.

This reduces manual calculation errors and ensures real-time accuracy.

### 6. ****Scalability and Modularity****

The structure supports additional future modules like tax computation, attendance tracking, leave management, or bonus history.

Each module is modular and maintains clean relationships with existing tables.

This technique-driven design ensures not only robust data storage but also makes the system powerful in generating operational and financial insights through dynamic SQL queries.

* System Configuration:
* **Database Technology:** MySQL
* **Supported Language:** SQL
* **Environment:** Any MySQL Client
* **Hardware/Software:** Supports all standard database server configurations.
* INPUT:

CREATE DATABASE SMS;

USE SMS;

CREATE TABLE Employee(

emp\_id INT PRIMARY KEY,

name VARCHAR(50) NOT NULL,

email VARCHAR(50) NOT NULL UNIQUE,

address VARCHAR(100) NOT NULL,

gender VARCHAR(20) NOT NULL CHECK (gender IN ('male', 'female', 'other')),

date\_of\_joining DATE NOT NULL

);

CREATE TABLE Department(

dept\_id INT PRIMARY KEY,

dept\_name VARCHAR(50) NOT NULL,

location VARCHAR(50) NOT NULL

);

CREATE TABLE Salary(

salary\_id INT PRIMARY KEY,

emp\_id INT NOT NULL,

basic\_salary DECIMAL(10, 2) NOT NULL CHECK (basic\_salary >= 0),

bonus DECIMAL(10, 2) NOT NULL CHECK (bonus >= 0),

deductions DECIMAL(10, 2) NOT NULL CHECK (deductions >= 0),

total\_salary DECIMAL(10, 2) GENERATED ALWAYS AS (basic\_salary + bonus - deductions) STORED,

FOREIGN KEY (emp\_id) REFERENCES Employee(emp\_id) ON DELETE CASCADE

);

CREATE TABLE Position(

pos\_id INT PRIMARY KEY,

emp\_id INT NOT NULL,

position\_name VARCHAR(50) NOT NULL,

FOREIGN KEY (emp\_id) REFERENCES Employee(emp\_id) ON DELETE CASCADE

);

CREATE TABLE EmployeeDepartment(

emp\_id INT NOT NULL,

dept\_id INT NOT NULL,

FOREIGN KEY (emp\_id) REFERENCES Employee(emp\_id) ON DELETE CASCADE,

FOREIGN KEY (dept\_id) REFERENCES Department(dept\_id) ON DELETE CASCADE,

PRIMARY KEY (emp\_id, dept\_id)

);

INSERT INTO Employee(emp\_id, name, email, address, gender, date\_of\_joining)

VALUES

(101, 'John Doe', 'john.doe@example.com', '1234 Elm Street, California', 'male', '2020-05-20'),

(102, 'Jane Smith', 'jane.smith@example.com', '5678 Oak Avenue, New York', 'female', '2019-07-15'),

(103, 'Mike Johnson', 'mike.johnson@example.com', '4321 Pine Road, Texas', 'male', '2021-01-10'),

(104, 'Alice Brown', 'alice.brown@example.com', '7890 Maple Street, California', 'female', '2022-03-01'),

(105, 'David Clark', 'david.clark@example.com', '1234 Birchwood, Florida', 'male', '2020-08-12'),

(106, 'Sarah Miller', 'sarah.miller@example.com', '8765 Cedar Blvd, Nevada', 'female', '2021-06-20'),

(107, 'Emily White', 'emily.white@example.com', '3456 Redwood Drive, Ohio', 'female', '2019-09-25'),

(108, 'Daniel King', 'daniel.king@example.com', '2345 Fir Avenue, Arizona', 'male', '2022-07-13'),

(109, 'Sophia Green', 'sophia.green@example.com', '5432 Maple Avenue, Texas', 'female', '2020-01-30'),

(110, 'James Adams', 'james.adams@example.com', '4567 Oak Street, California', 'male', '2021-02-12'),

(111, 'Benjamin Scott', 'benjamin.scott@example.com', '8901 Pine Lane, Georgia', 'male', '2021-11-01'),

(112, 'Charlotte Carter', 'charlotte.carter@example.com', '1357 Oak Blvd, Michigan', 'female', '2022-05-25');

INSERT INTO Department(dept\_id, dept\_name, location)

VALUES

(11, 'Human Resources', 'California'),

(12, 'Finance', 'New York'),

(13, 'Engineering', 'Texas');

INSERT INTO Salary(salary\_id, emp\_id, basic\_salary, bonus, deductions)

VALUES

(201, 101, 5000, 500, 200),

(202, 102, 6000, 600, 250),

(203, 103, 7000, 700, 300),

(204, 104, 6500, 650, 150),

(205, 105, 7500, 750, 100),

(206, 106, 8000, 800, 350),

(207, 107, 5500, 500, 200),

(208, 108, 6700, 670, 150),

(209, 109, 7200, 720, 250),

(210, 110, 7800, 780, 300),

(211, 111, 8200, 820, 400),

(212, 112, 8800, 880, 450);

INSERT INTO Position(pos\_id, emp\_id, position\_name)

VALUES

(301, 101, 'HR Manager'),

(302, 102, 'Financial Analyst'),

(303, 103, 'Software Engineer'),

(304, 104, 'Marketing Executive'),

(305, 105, 'HR Assistant'),

(306, 106, 'Financial Manager'),

(307, 107, 'Content Writer'),

(308, 108, 'Software Developer'),

(309, 109, 'Accountant'),

(310, 110, 'Financial Planner'),

(311, 111, 'Web Developer'),

(312, 112, 'Project Manager');

INSERT INTO EmployeeDepartment(emp\_id, dept\_id)

VALUES

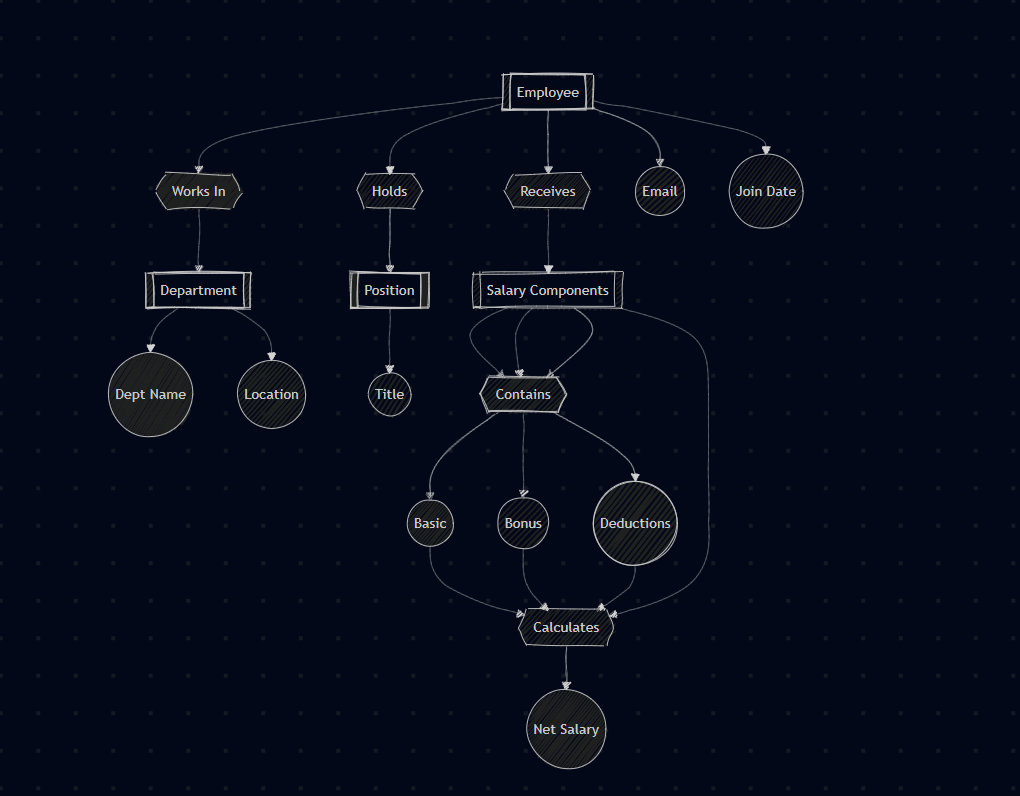
(101, 11), (102, 12), (103, 13),

(104, 13), (105, 11), (106, 12),

(107, 11), (108, 13), (109, 12),

(110, 13), (111, 12), (112, 11);

* ER DIAGRAM:



* TABLE REALTION:

| **Table 1** | **Table 2** | **Relationship Type** |
| --- | --- | --- |
| Employee | Salary | One-to-One (each employee has one salary record) |
| Employee | Position | One-to-One (each employee holds one position) |
| Employee | EmployeeDepartment | One-to-Many (employee can be assigned to many departments) |
| Department | EmployeeDepartment | One-to-Many (a department has many employees) |
| Employee | Salary | One-to-One (linked via emp\_id as FK) |

* TABULAR FORMAT:

### Employee

emp\_id (INT, PK)

name (VARCHAR)

email (VARCHAR, UNIQUE)

address (VARCHAR)

gender (VARCHAR)

date\_of\_joining (DATE)

### Department

dept\_id (INT, PK)

dept\_name (VARCHAR)

location (VARCHAR)

### Salary

salary\_id (INT, PK)

emp\_id (INT, FK → Employee.emp\_id)

basic\_salary (DECIMAL)

bonus (DECIMAL)

deductions (DECIMAL)

total\_salary (DECIMAL, generated)

### Position

pos\_id (INT, PK)

emp\_id (INT, FK → Employee.emp\_id)

position\_name (VARCHAR)

### EmployeeDepartment

emp\_id (INT, FK → Employee.emp\_id)

dept\_id (INT, FK → Department.dept\_id)

**PK:** (emp\_id, dept\_id)

* SQL QUERIES WITH OUTPUT (at least 15 ):

### 1. ****Get all employee names and their total salary****

sql

Copy code

SELECT e.name, s.total\_salaryFROM Employee eJOIN Salary s ON e.emp\_id = s.emp\_id;

### 2. ****Update salary of an employee****

sql

Copy code

UPDATE SalarySET basic\_salary = 5500, bonus = 550, deductions = 100WHERE emp\_id = 101;

### 3. ****Delete an employee and cascade related records****

sql

Copy code

DELETE FROM EmployeeWHERE email = 'alice.brown@example.com';

### 4. ****Select employee, department, and total salary****

sql

Copy code

SELECT e.name, d.dept\_name, s.total\_salaryFROM Employee eJOIN EmployeeDepartment ed ON e.emp\_id = ed.emp\_idJOIN Department d ON ed.dept\_id = d.dept\_idJOIN Salary s ON e.emp\_id = s.emp\_id;

### 5. ****Select employee names and their positions****

sql

Copy code

SELECT e.name, p.position\_nameFROM Employee eJOIN Position p ON e.emp\_id = p.emp\_id;

### 6. ****Total salary of all employees per department****

sql

Copy code

SELECT d.dept\_name, SUM(s.total\_salary) AS total\_salaryFROM Department dJOIN EmployeeDepartment ed ON d.dept\_id = ed.dept\_idJOIN Employee e ON ed.emp\_id = e.emp\_idJOIN Salary s ON e.emp\_id = s.emp\_idGROUP BY d.dept\_name;

### 7. ****Employees who joined after 2021****

sql

Copy code

SELECT name, date\_of\_joiningFROM EmployeeWHERE date\_of\_joining > '2021-01-01';

### 8. ****List all female employees****

sql

Copy code

SELECT nameFROM EmployeeWHERE gender = 'female';

### 9. ****Count of employees per department****

sql

Copy code

SELECT d.dept\_name, COUNT(\*) AS total\_employeesFROM Department dJOIN EmployeeDepartment ed ON d.dept\_id = ed.dept\_idGROUP BY d.dept\_name;

### 10. ****Top 3 employees by salary****

sql

Copy code

SELECT e.name, s.total\_salaryFROM Employee eJOIN Salary s ON e.emp\_id = s.emp\_idORDER BY s.total\_salary DESC

LIMIT 3;

### 11. ****Find employees whose salary is more than 7000****

sql

Copy code

SELECT e.name, s.total\_salaryFROM Employee eJOIN Salary s ON e.emp\_id = s.emp\_idWHERE s.total\_salary > 7000;

### 12. ****Find employees working in California****

sql

Copy code

SELECT nameFROM EmployeeWHERE address LIKE '%California%';

### 13. ****List employee names and their joining year****

sql

Copy code

SELECT name, YEAR(date\_of\_joining) AS joining\_yearFROM Employee;

### 14. ****Average salary in each department****

sql

Copy code

SELECT d.dept\_name, AVG(s.total\_salary) AS avg\_salaryFROM Department dJOIN EmployeeDepartment ed ON d.dept\_id = ed.dept\_idJOIN Salary s ON ed.emp\_id = s.emp\_idGROUP BY d.dept\_name;

### 15. ****Find departments without any employees (if any)****

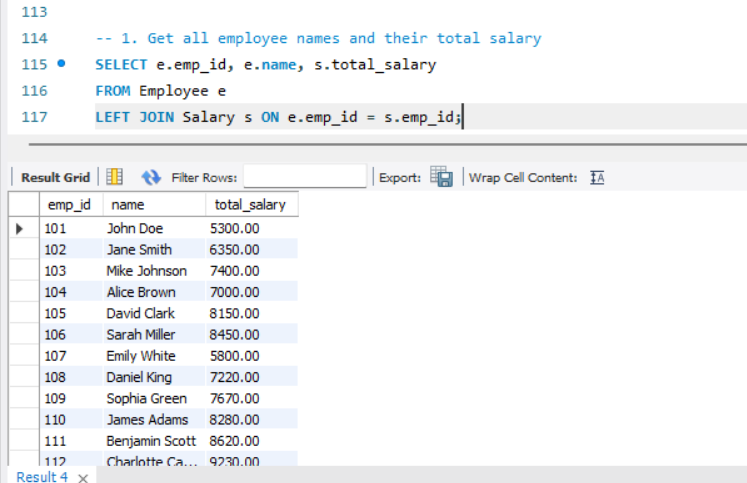
sql

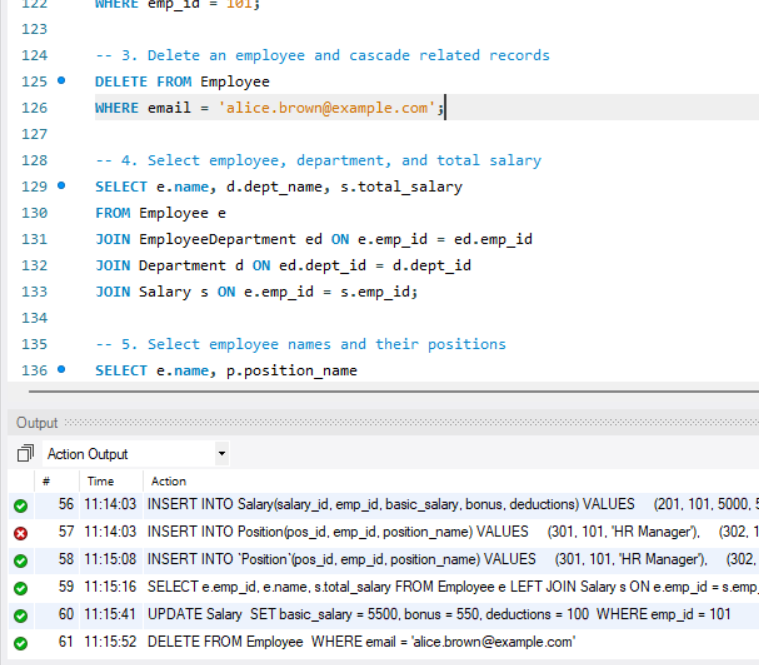
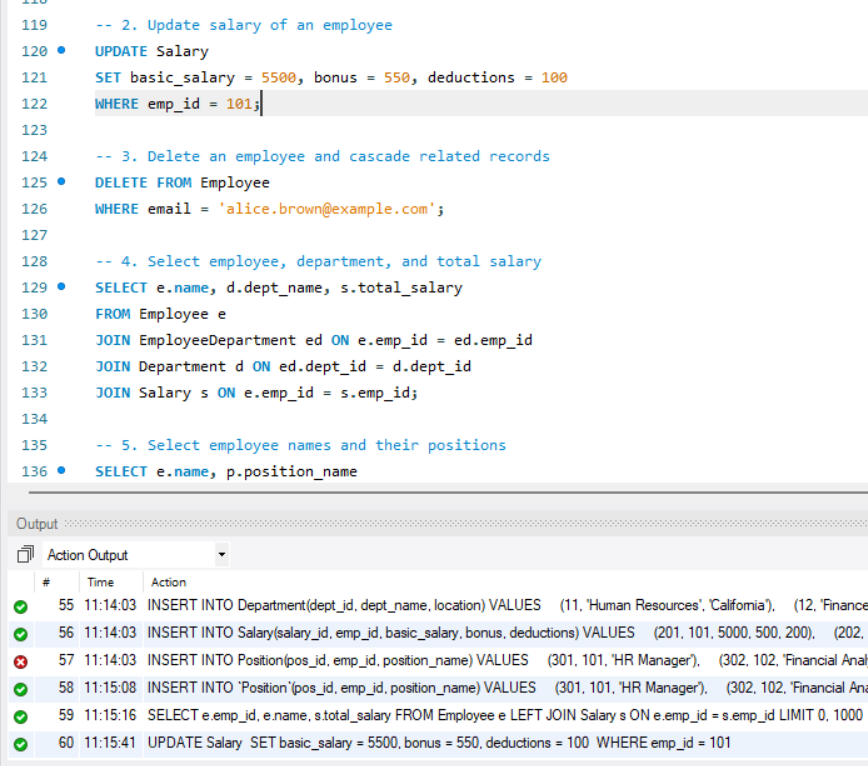
Copy code

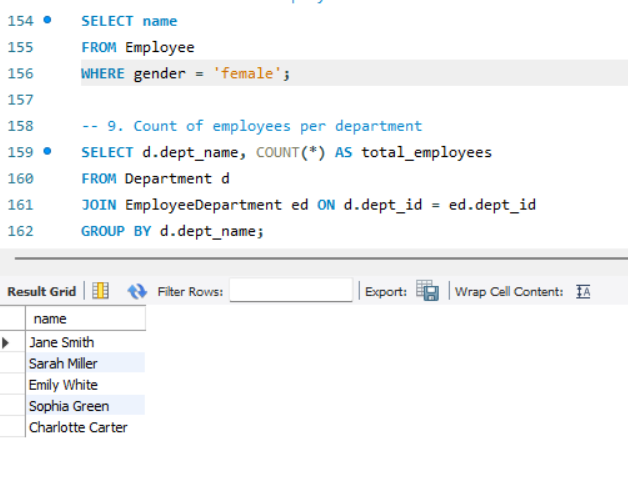
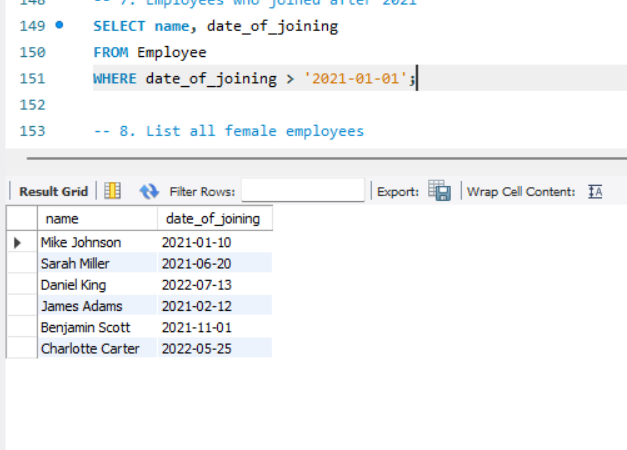
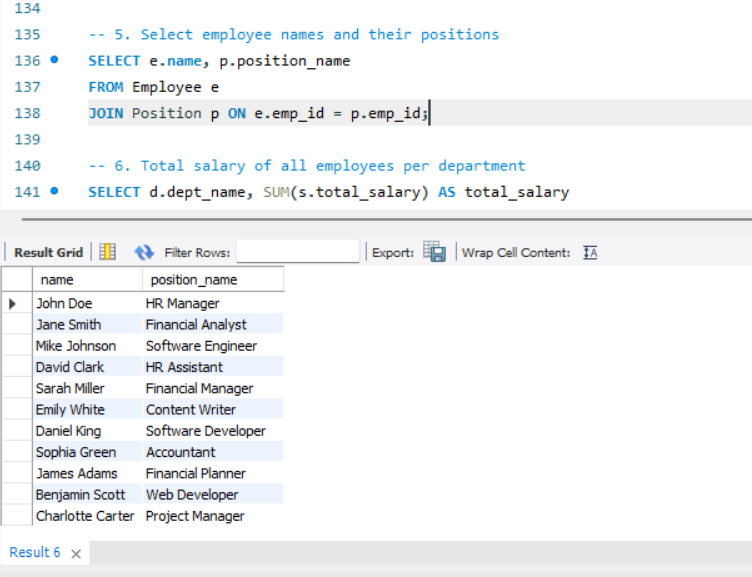
SELECT dept\_nameFROM DepartmentWHERE dept\_id NOT IN (

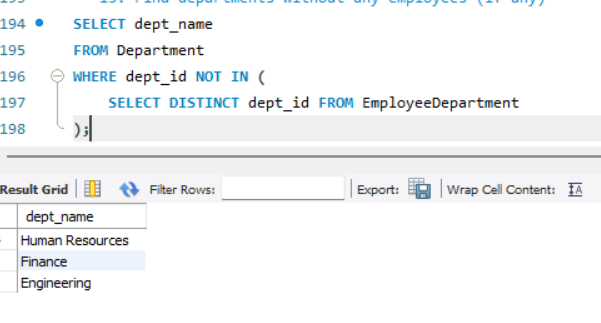
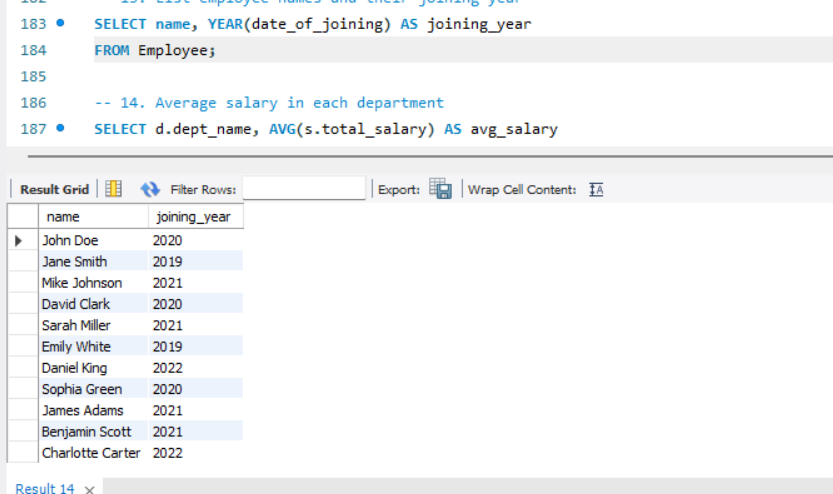
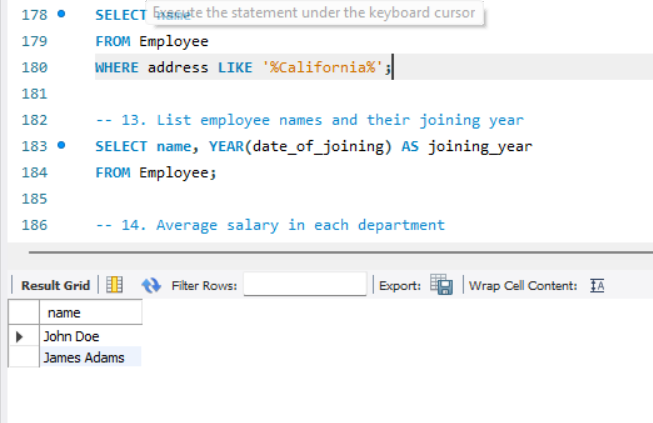
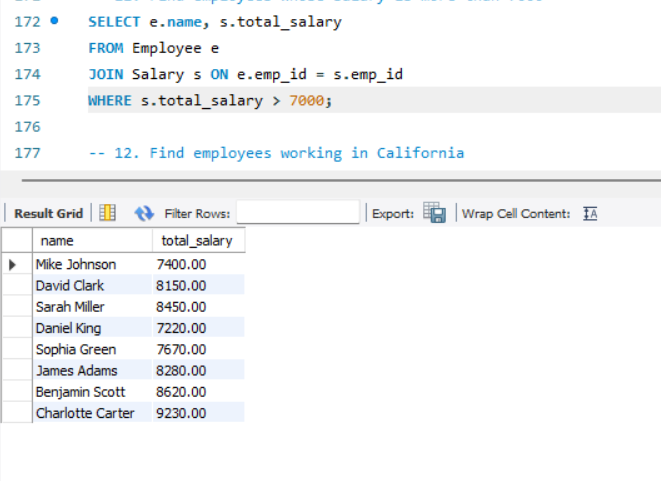
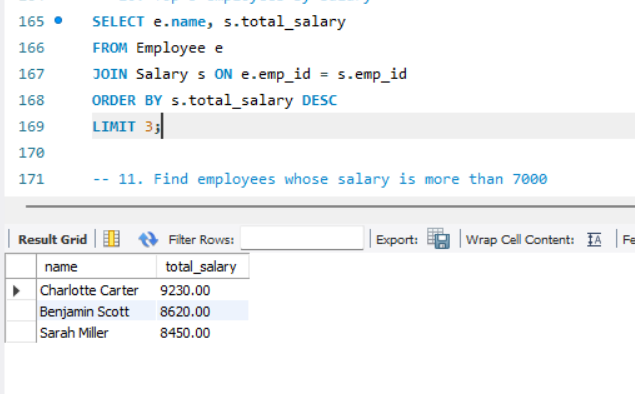
SELECT DISTINCT dept\_id FROM EmployeeDepartment

);









* SUMMARY:

The **Salary Management System (SMS)** is a well-structured and normalized relational database system designed to efficiently manage salary-related and organizational information for employees. It plays a critical role in automating tasks that were previously performed manually, such as calculating total salaries, assigning positions, managing departments, and ensuring employee data consistency.

Built using **MySQL**, the SMS is composed of multiple interlinked tables—such as Employee, Salary, Department, Position, and EmployeeDepartment—that form the backbone of the system. Each table is connected through primary and foreign key constraints to ensure data consistency and relational integrity. The normalization applied throughout the design ensures minimal redundancy and improved data retrieval performance.

This system supports a wide range of core functionalities including:

Accurate **salary calculation**, using basic pay, bonuses, and deductions.

Assignment of **employees to departments** for structured organizational management.

Tracking of **employee roles and positions** within the company hierarchy.

Generation of **analytical reports** like average department salary, top earners, or departmental payroll statistics.

Enforcement of **data integrity rules** using constraints such as CHECK, UNIQUE, and cascading foreign keys.

Furthermore, the SMS database is optimized for scalability and extensibility. Additional modules such as attendance tracking, taxation records, leave management, and payroll distribution can be seamlessly integrated in the future. The use of SQL queries with aggregation and joins ensures that the system not only stores data but also provides powerful insights into organizational operations.

In summary, the SMS ensures smooth operation of payroll and HR activities, supports organizational transparency, and allows decision-makers to act on reliable and timely data.

* CONCLUSION:

The **Salary Management System (SMS) Database** provides a comprehensive solution for handling employee information, salary structure, departmental allocation, and positional responsibilities in a structured and efficient manner. It showcases how an organization can streamline HR and financial operations using the principles of **relational database management systems (RDBMS)**.

By utilizing a normalized schema, the system avoids data duplication and ensures that all employee-related records are properly linked and synchronized across various tables. The inclusion of CHECK constraints and FOREIGN KEY relationships ensures that only valid, consistent, and complete data is stored in the system. Moreover, the use of computed fields like total\_salary in the Salary table demonstrates how automation can be used to reduce manual calculations and potential errors.

This project also demonstrates the power of **SQL** for real-world applications. Through a variety of **SELECT**, **JOIN**, **AGGREGATE**, **UPDATE**, **DELETE**, and **INSERT** queries, the system enables flexible data handling and dynamic reporting—ranging from individual employee pay summaries to department-wide payroll analytics.

Key benefits realized through this system include:

**Efficiency**: Reduces manual workload by automating salary and departmental record management.

**Accuracy**: Eliminates discrepancies by enforcing strict validation rules and constraints.

**Scalability**: Supports future enhancements without redesigning the existing structure.

**Integrity**: Maintains relational integrity and ensures that linked data remains consistent.

In conclusion, the Salary Management System Database not only fulfills the fundamental requirements of employee salary management but also lays a solid foundation for more advanced HR and financial systems. It is a reliable, secure, and intelligent system that improves organizational productivity, transparency, and decision-making.