Course Project: EEMS - Employment Management System

- Project Goal and Architecture

The primary goal of this project is to design and implement a non-framework-dependent N-Tier Java application using pure JDBC for data persistence. Students must demonstrate proficiency in translating a real-world domain model into a structured, layered, and maintainable object-oriented system. Your application must strictly adhere to the following n-tier structure:

1. **Domain Model Layer:** Java classes representing the entities (Employee, Department, etc.). These contain only data and business logic related to the entity itself.
2. **Business Logic Layer (Service):** Contains all the core application logic, validation rules, and complex calculations. It utilizes the DAL to retrieve/store data.
3. **Data Access Layer (Repository):** Handles all persistence logic (e.g., connecting to the database, executing SQL via JDBC, mapping result sets to java classes). **No business logic is allowed here.**
4. **Presentation Layer (Controller/CLI):** This layer will be controllers for each domain providing the CRUD operations and the required tasks mentioned below.

*Note: This project is purely backend, there are no requirement for working on a user interface.

- Problem Statement

The Employment Management System (EEMS) is intended to be the single source of truth for all resource and operational tracking within the company. Your first task is to read this narrative carefully and extract the four core domain entities required to build the system.

The company is structured into distinct groups, each managed by a leadership team. These organizational units (departments) have a specific name, a fixed location (building or city), and an annual budget. These units are responsible for hosting and executing multiple operational tasks.

Every person working at the company is considered part of the workforce. For each individual, the system must record their full name, current title (e.g., Manager, Specialist), hire date, and current salary. Crucially, every single member of the workforce must be permanently assigned to one, and only one, organizational unit.

Operational tasks are structured as defined projects. Each project must have a name, detailed description, specific start and end dates, and a defined budget. Projects can be in various stages, tracked by their status (e.g., 'Active', 'Completed'). These projects are typically managed or hosted collaboratively by multiple organizational units. Furthermore, the workforce works on these projects. Any single person can be assigned to multiple projects simultaneously, and naturally, any single project will have many people working on it. The system must track the percentage of time a worker is allocated to each specific project.

Finally, the company tracks external relationships via a registry of associated clients. Each client is defined by their name, the industry they operate in, and the primary contact person's name and contact information (phone and email). A single project can be tied to several different clients, and conversely, a single client may sponsor or be associated with numerous different projects across the company. The system must maintain these links accurately.

The system must ensure full CRUD (Create, Read, Update, Delete) functionality for all extracted domain entities and maintain strict referential integrity based on the relationships described above.

- Required Functionality

Core Operations (CRUD)

Students must implement the **Create, Read, Update, and Delete (CRUD)** operations for all primary entities.

- **Create:** Insert a new record into the database.
- **Read:** Retrieve a single record by ID, and retrieve a list of all records.
- **Update:** Modify an existing record's attributes.
- **Delete:** Remove a record. Implement appropriate cascade/referential integrity handling (e.g., what happens to employees if a department is deleted? Throw an exception).

Specific Business Logic Tasks

In addition to CRUD, the Business Logic Layer must implement the following four domain-specific methods:

Task 1: Cost Calculation - `calculateProjectHRCost(int projectId)`

This method must calculate the projected human resource cost for a given project based on the duration, employee salaries, and their allocated time.

1. Determine the project duration in **months** (rounded up to the next full month).
2. For every employee assigned to the project, calculate their weighted cost:

$$Cost_{Employee} = \frac{Salary}{12} \times \text{Duration (Months)} \times \frac{\text{Allocation Percentage}}{100}$$

3. The result is the sum of Cost of Employee for all assigned employees.

Task 2: Department Project Report - `getProjectsByDepartment(int departmentId, String sortBy)`

Retrieve a list of all **Active** projects associated with a specific department. The list must be sorted by a specified field (e.g., `project_budget`, `end_date`).

Task 3: High-Value Client Identification - `findClientsByUpcomingProjectDeadline(int daysUntilDeadline)`

Find and return a list of all **Client** objects that are linked to one or more projects scheduled to end within the specified number of days from the current date.

Task 4: Employee Transfer Transaction - `transferEmployeeToDepartment(int employeeId, int newDepartmentId)`

Perform a transaction to update an employee's department ID. This requires interacting with both the `Employee` and `Department` records. Implement validation to ensure the transfer is possible before committing the changes.

- Required Deliverables (Design Phase)

The following design artifacts must be submitted **before** the code implementation:

1. Domain Class Definition

A list of all necessary classes, including attributes, data types, and brief justifications for each class.

2. Class Diagram (UML)

A complete UML Class Diagram showing all Domain Model classes, their attributes, methods (including getters/setters), and the full multiplicity (1:1, 1:N, N:M) of their relationships.

3. Use Case and Sequence Diagrams

1. **Use Case Diagram:** A diagram illustrating the user roles and system functions for the entire EEMS (e.g., "Manage Employees," "Calculate Project Cost").
2. **Sequence Diagram:** A detailed sequence diagram illustrating the method calls and component interactions specifically for **Task 1:** `calculateProjectHRCost(int projectId)`, showing the flow from the Controller through the Service Layer, into the Data access Layer, and interaction with the database.

4. Database Schema Design and Sample Data

Provide the SQL `CREATE TABLE` statements for the relational database schema that supports the Domain Model. This must include all primary keys (PK), foreign keys (FK), junction tables, and data types.
The schema must include junction tables to handle the required Many-to-Many relationships:

- Example: `Employee_Project` (M:M between Employee and Project)

Following the schema, provide **five (5) sample SQL `INSERT` statements** for each table to fully populate the system with realistic mock data.

- Submission Guidelines

- **Code Implementation:** Java code using the provided N-Tier structure. All data persistence must use standard **JDBC**—no ORMs (like Hibernate, JPA) or frameworks (like Spring) are required
- **Testing:** Include a simple testing class (with a `main` method) that demonstrates successful execution of all required CRUD and business logic tasks.
- **Build DB tables script:** Provide a sql script that will build all tables and add some sample data.
- **AI/Assistant Policy:** The use of AI tools (like chatGPT) or other coding assistants is permitted and encouraged, provided that **all submitted work is understood and can be explained verbally** during the presentation phase. Simply copy-pasting code without comprehension will result in deductions.

- **Presentation:** Details regarding the mandatory project presentation will be shared later. Be prepared to explain your design choices (UML, Schema) and demonstrate your code's functionality.