**Day 5 – Attendance Feature Implementation**

**What I Accomplished Today**

As part of my HRMS project development, I successfully implemented the attendance tracking feature. This involved creating the database structure, model class, and data access operations.

**1. Database Table Setup**

I created the attendance table in my database with the following structure:

CREATE TABLE attendance (

id INT PRIMARY KEY AUTO\_INCREMENT,

employee\_id INT NOT NULL,

date DATE NOT NULL,

status VARCHAR(20) NOT NULL,

FOREIGN KEY (employee\_id) REFERENCES employees(id)

);

**Note**: In my DAO code, I used date as the column name in the SQL queries, but my Java model uses day as the field name. This works because I handle the mapping in my DAO methods.

**2. Attendance Model Class**

I created the Attendance.java class in the com.hrms.model package:

package com.hrms.model;

import java.time.LocalDate;

public class Attendance {

private int id;

private int employeeId;

private LocalDate day;

private String status; // Present, Absent, Leave, etc.

// Constructors

public Attendance() {}

public Attendance(int id, int employeeId, LocalDate day, String status) {

this.id = id;

this.employeeId = employeeId;

this.day = day;

this.status = status;

}

// Getters & Setters

public int getId() { return id; }

public void setId(int id) { this.id = id; }

public int getEmployeeId() { return employeeId; }

public void setEmployeeId(int employeeId) { this.employeeId = employeeId; }

public LocalDate getDay() { return day; }

public void setDay(LocalDate day) { this.day = day; }

public String getStatus() { return status; }

public void setStatus(String status) { this.status = status; }

@Override

public String toString() {

return "Attendance{id=" + id + ", empId=" + employeeId +

", day=" + day + ", status='" + status + "'}";

}

}

**Key Design Decisions:**

* Used LocalDate for better date handling in Java 8+
* Kept the field name as day (though database column is date)
* Added a comprehensive toString() method for debugging
* Status field allows flexible attendance states

**3. AttendanceDAO Implementation**

I created the AttendanceDAO.java class in the com.hrms.model.dao package with three main operations:

package com.hrms.model.dao;

import com.hrms.model.Attendance;

import java.sql.\*;

import java.util.ArrayList;

import java.util.List;

public class AttendanceDAO {

// Add attendance

public void addAttendance(Attendance a) {

String sql = "INSERT INTO attendance (employee\_id, date, status) VALUES (?, ?, ?)";

try (Connection conn = DBConnection.getConnection();

PreparedStatement ps = conn.prepareStatement(sql)) {

ps.setInt(1, a.getEmployeeId());

ps.setDate(2, Date.valueOf(a.getDay()));

ps.setString(3, a.getStatus());

ps.executeUpdate();

System.out.println(" Attendance marked for empId=" + a.getEmployeeId() +

" on " + a.getDay());

} catch (SQLException e) {

System.err.println(" Error marking attendance: " + e.getMessage());

}

}

// Get attendance by employee

public List<Attendance> getAttendanceByEmployee(int empId) {

List<Attendance> list = new ArrayList<>();

String sql = "SELECT \* FROM attendance WHERE employee\_id = ? ORDER BY date DESC";

try (Connection conn = DBConnection.getConnection();

PreparedStatement ps = conn.prepareStatement(sql)) {

ps.setInt(1, empId);

try (ResultSet rs = ps.executeQuery()) {

while (rs.next()) {

list.add(new Attendance(

rs.getInt("id"),

rs.getInt("employee\_id"),

rs.getDate("date").toLocalDate(),

rs.getString("status")

));

}

}

} catch (SQLException e) {

System.err.println(" Error fetching attendance: " + e.getMessage());

}

return list;

}

// Get all attendance records

public List<Attendance> getAllAttendance() {

List<Attendance> list = new ArrayList<>();

String sql = "SELECT \* FROM attendance ORDER BY date DESC";

try (Connection conn = DBConnection.getConnection();

Statement st = conn.createStatement();

ResultSet rs = st.executeQuery(sql)) {

while (rs.next()) {

list.add(new Attendance(

rs.getInt("id"),

rs.getInt("employee\_id"),

rs.getDate("date").toLocalDate(),

rs.getString("status")

));

}

} catch (SQLException e) {

System.err.println(" Error fetching all attendance: " + e.getMessage());

}

return list;

}

}

**4. What Each Method Does**

**addAttendance(Attendance a)**

* **Purpose**: Adds a new attendance record to the database
* **Input**: Attendance object with employee ID, date, and status
* **Key Learning**: Used Date.valueOf(a.getDay()) to convert LocalDate to SQL Date
* **Output**: Prints success message or error if something goes wrong

**getAttendanceByEmployee(int empId)**

* **Purpose**: Retrieves all attendance records for a specific employee
* **Input**: Employee ID
* **Features**:
  + Orders results by date (newest first)
  + Uses PreparedStatement to prevent SQL injection
  + Converts SQL Date back to LocalDate using .toLocalDate()
* **Output**: List of Attendance objects

**getAllAttendance()**

* **Purpose**: Gets all attendance records from the database
* **Features**:
  + Uses simple Statement (no parameters needed)
  + Orders by date descending
  + Good for admin/HR to see company-wide attendance
* **Output**: List of all Attendance objects

**5. Technical Implementation Details**

**Database Connection**

* Used the existing DBConnection.getConnection() method
* Implemented try-with-resources for automatic resource management
* Added proper exception handling with user-friendly error messages

**Data Type Conversions**

* **LocalDate to SQL Date**: Date.valueOf(localDate)
* **SQL Date to LocalDate**: sqlDate.toLocalDate()
* This ensures compatibility between Java 8+ date API and SQL dates

**Error Handling**

* Added console output with ✅ and ❌ emojis for clear feedback
* Used System.err.println() for errors
* Try-catch blocks around all database operations

**6. What I Learned**

1. **Date Handling**: Managing the difference between Java LocalDate and SQL DATE types
2. **PreparedStatement**: Using parameterized queries for security and performance
3. **Resource Management**: Try-with-resources ensures connections are properly closed
4. **Error Handling**: Importance of catching and displaying meaningful error messages
5. **Data Mapping**: Converting between database rows and Java objects