JAVA CASE STUDY

Name: Kavin Kaarthik

Batch: 03

SQL SCHEMA

create table Project (

id INT PRIMARY KEY AUTO\_INCREMENT,

project\_name VARCHAR(255) NOT NULL,

description TEXT,

start\_date DATE NOT NULL,

status ENUM('started', 'dev', 'build', 'test', 'deployed') NOT NULL

);

create table Employee (

id INT primary key auto\_increment,

name VARCHAR(255) not null,

designation VARCHAR(100) not null,

gender ENUM('Male', 'Female', 'Other') not null,

salary DECIMAL(10,2) not null,

project\_id INT,

FOREIGN KEY (project\_id) references Project(id) on delete set null);

create table Task (

task\_id INT primary key auto\_increment,

task\_name VARCHAR(255) not null,

project\_id INT,

employee\_id INT,

status ENUM('Assigned', 'Started', 'Completed') not null,

FOREIGN KEY (project\_id) REFERENCES Project(id) on delete cascade,

FOREIGN KEY (employee\_id) REFERENCES Employee(id) on delete set null);

INSERT INTO Project (project\_name, description, start\_date, status) VALUES

('Website Redesign', 'Revamping the corporate website with a modern design.', '2024-01-15', 'started'),

('Mobile App Development', 'Developing a mobile app for e-commerce features.', '2024-02-01', 'dev'),

('Inventory System', 'Building an internal inventory management system.', '2024-03-10', 'build'),

('Customer Feedback Portal', 'Platform for collecting and analyzing customer feedback.', '2024-01-25', 'test'),

('AI Chatbot Integration', 'Integrating AI chatbot into the customer support system.', '2024-04-01', 'deployed');

INSERT INTO Employee (name, designation, gender, salary, project\_id) VALUES

('Kavin Kaarthik', 'Software Engineer', 'Male', 65000.00, 1),

('Sneha', 'Frontend Developer', 'Female', 60000.00, 1),

('Arjun', 'Backend Developer', 'Male', 70000.00, 2),

('Priya Menon', 'Project Manager', 'Female', 90000.00, 3),

('Rahul Das', 'QA Engineer', 'Male', 55000.00, 4),

('Anjali Rao', 'Data Scientist', 'Female', 85000.00, 5),

('Sam Wilson', 'DevOps Engineer', 'Other', 75000.00, 3);

INSERT INTO Task (task\_name, project\_id, employee\_id, status) VALUES

('Create wireframes', 1, 2, 'Completed'),

('Implement backend APIs', 2, 3, 'Started'),

('Database schema design', 3, 4, 'Assigned'),

('Write test cases', 4, 5, 'Started'),

('Deploy application', 5, 6, 'Completed'),

('CI/CD setup', 3, 7, 'Started'),

('UI implementation', 1, 1, 'Assigned'),

('Bug fixing', 2, 3, 'Assigned');

JAVA IMPLEMETATION

package app;

import dao.ProjectRepositoryImpl;

import util.DBConnUtil;

import entity.Employee;

import entity.Project;

import entity.Task;

import java.util.Scanner;

public class ProjectApp {

public static void main(String[] args) {

// Initialize DB connection

DBConnUtil.getConnection("db.properties");

ProjectRepositoryImpl repo = new ProjectRepositoryImpl();

try (Scanner sc = new Scanner(System.in)) {

int choice;

do {

System.out.println("\n--- Project Management Menu ---");

System.out.println("1. Add Employee");

System.out.println("2. Add Project");

System.out.println("3. Add Task");

System.out.println("4. Assign Project to Employee");

System.out.println("5. Assign Task in Project to Employee");

System.out.println("6. Delete Employee");

System.out.println("7. Delete Task");

System.out.println("8. List All Tasks for Employee in Project");

System.out.println("9. Exit");

choice = sc.nextInt();

sc.nextLine(); // Consume newline

try {

switch (choice) {

case 1: // Add Employee

System.out.print("Enter employee name: ");

String name = sc.nextLine();

System.out.print("Enter designation: ");

String designation = sc.nextLine();

System.out.print("Enter gender (Male/Female/Other): ");

String gender = sc.nextLine();

System.out.print("Enter salary: ");

double salary = sc.nextDouble();

System.out.print("Enter project ID (optional, 0 for none): ");

int projectId = sc.nextInt();

sc.nextLine(); // Consume newline

Employee emp = new Employee(name, designation, gender, salary, projectId == 0 ? null : projectId);

if (repo.createEmployee(emp)) {

System.out.println("Employee added successfully!");

} else {

System.out.println("Failed to add employee.");

}

break;

case 2: // Add Project

System.out.print("Enter project name: ");

String projectName = sc.nextLine();

System.out.print("Enter project description: ");

String description = sc.nextLine();

System.out.print("Enter project start date (yyyy-mm-dd): ");

String startDate = sc.nextLine();

System.out.print("Enter project status (started/dev/build/test/deployed): ");

String status = sc.nextLine();

Project pj = new Project(projectName, description, startDate, status);

if (repo.createProject(pj)) {

System.out.println("Project added successfully!");

} else {

System.out.println("Failed to add project.");

}

break;

case 3: // Add Task

System.out.print("Enter task name: ");

String taskName = sc.nextLine();

System.out.print("Enter project ID: ");

int taskProjectId = sc.nextInt();

System.out.print("Enter employee ID: ");

int taskEmployeeId = sc.nextInt();

sc.nextLine(); // Consume newline

System.out.print("Enter task status (Assigned/Started/Completed): ");

String taskStatus = sc.nextLine();

Task tk = new Task(taskName, taskProjectId, taskEmployeeId, taskStatus);

if (repo.createTask(tk)) {

System.out.println("Task added successfully!");

} else {

System.out.println("Failed to add task.");

}

break;

case 4: // Assign Project to Employee

System.out.print("Enter employee ID: ");

int empId = sc.nextInt();

System.out.print("Enter project ID: ");

int projectID = sc.nextInt();

if (repo.assignProjectToEmployee(projectID, empId)) {

System.out.println("Project assigned to employee successfully!");

} else {

System.out.println("Failed to assign project.");

}

break;

case 5: // Assign Task to Employee

System.out.print("Enter task ID: ");

int taskId = sc.nextInt();

System.out.print("Enter project ID: ");

int projId = sc.nextInt();

System.out.print("Enter employee ID: ");

int empID = sc.nextInt();

if (repo.assignTaskInProjectToEmployee(taskId, projId, empID)) {

System.out.println("Task assigned to employee successfully!");

} else {

System.out.println("Failed to assign task.");

}

break;

case 6: // Delete Employee

System.out.print("Enter employee ID: ");

int deleteEmpId = sc.nextInt();

if (repo.deleteEmployee(deleteEmpId)) {

System.out.println("Employee deleted successfully!");

} else {

System.out.println("Failed to delete employee.");

}

break;

case 7: // Delete Task

System.out.print("Enter task ID: ");

int deleteTaskId = sc.nextInt();

if (repo.deleteTask(deleteTaskId)) {

System.out.println("Task deleted successfully!");

} else {

System.out.println("Failed to delete task.");

}

break;

case 8: // List all tasks for employee in project

System.out.print("Enter employee ID: ");

int empIdForTasks = sc.nextInt();

System.out.print("Enter project ID: ");

int projIdForTasks = sc.nextInt();

var tasks = repo.getAllTasks(empIdForTasks, projIdForTasks);

tasks.forEach(task -> System.out.println(task.getTaskName()));

break;

case 9:

System.out.println("Exiting...");

break;

default:

System.out.println("Invalid choice. Please try again.");

break;

}

} catch (Exception e) {

System.out.println("An error occurred: " + e.getMessage());

e.printStackTrace();

}

} while (choice != 9);

} catch (Exception e) {

System.out.println("Unexpected error: " + e.getMessage());

e.printStackTrace();

} finally {

DBConnUtil.closeConnection();

System.out.println("Application terminated. DB connection closed.");

}

}

}

package dao;

import entity.Employee;

import entity.Project;

import entity.Task;

import java.util.List;

public interface IProjectRepository {

boolean createEmployee(Employee emp);

boolean createProject(Project pj);

boolean createTask(Task tk);

boolean assignProjectToEmployee(int projectId, int employeeId);

boolean assignTaskInProjectToEmployee(int taskId, int projectId, int employeeId);

boolean deleteEmployee(int userId);

boolean deleteProject(int projectId);

boolean deleteTask(int taskId); // Add this line

List<Task> getAllTasks(int empId, int projectId);

List<Project> searchProjectsByEmployeeId(int empId);

List<Task> searchTasksByEmployeeId(int empId);

List<Task> searchTasksByProjectId(int projectId);

}

package dao;

import entity.Employee;

import entity.Project;

import entity.Task;

import util.DBConnUtil;

import java.sql.\*;

import java.util.ArrayList;

import java.util.List;

import myexceptions.EmployeeNotFoundException;

import myexceptions.ProjectNotFoundException;

public class ProjectRepositoryImpl implements IProjectRepository {

private final Connection connection;

public ProjectRepositoryImpl() {

this.connection = DBConnUtil.*getConnection*("db.properties");

}

*@Override*

public boolean createEmployee(Employee emp) {

String sql = "INSERT INTO Employee (name, designation, gender, salary, project\_id) VALUES (?, ?, ?, ?, ?)";

try (PreparedStatement statement = connection.prepareStatement(sql)) {

statement.setString(1, emp.getName());

statement.setString(2, emp.getDesignation());

statement.setString(3, emp.getGender());

statement.setDouble(4, emp.getSalary());

statement.setObject(5, emp.getProjectId(), Types.***INTEGER***);

return statement.executeUpdate() > 0;

} catch (SQLException e) {

e.printStackTrace();

return false;

}

}

*@Override*

public boolean createProject(Project pj) {

String sql = "INSERT INTO Project (project\_name, description, start\_date, status) VALUES (?, ?, ?, ?)";

try (PreparedStatement statement = connection.prepareStatement(sql)) {

statement.setString(1, pj.getProjectName());

statement.setString(2, pj.getDescription());

statement.setString(3, pj.getStartDate());

statement.setString(4, pj.getStatus());

return statement.executeUpdate() > 0;

} catch (SQLException e) {

e.printStackTrace();

return false;

}

}

*@Override*

public boolean createTask(Task tk) {

String sql = "INSERT INTO Task (task\_name, project\_id, employee\_id, status) VALUES (?, ?, ?, ?)";

try (PreparedStatement statement = connection.prepareStatement(sql)) {

statement.setString(1, tk.getTaskName());

statement.setInt(2, tk.getProjectId());

statement.setInt(3, tk.getEmployeeId());

statement.setString(4, tk.getStatus());

return statement.executeUpdate() > 0;

} catch (SQLException e) {

e.printStackTrace();

return false;

}

}

*@Override*

public boolean assignProjectToEmployee(int projectId, int employeeId) {

String sql = "UPDATE Employee SET project\_id = ? WHERE id = ?";

try (PreparedStatement statement = connection.prepareStatement(sql)) {

statement.setInt(1, projectId);

statement.setInt(2, employeeId);

return statement.executeUpdate() > 0;

} catch (SQLException e) {

e.printStackTrace();

return false;

}

}

*@Override*

public boolean assignTaskInProjectToEmployee(int taskId, int projectId, int employeeId) {

String sql = "UPDATE Task SET employee\_id = ? WHERE task\_id = ? AND project\_id = ?";

try (PreparedStatement statement = connection.prepareStatement(sql)) {

statement.setInt(1, employeeId);

statement.setInt(2, taskId);

statement.setInt(3, projectId);

return statement.executeUpdate() > 0;

} catch (SQLException e) {

e.printStackTrace();

return false;

}

}

*@Override*

public boolean deleteEmployee(int userId) {

String sql = "DELETE FROM Employee WHERE id = ?";

try (PreparedStatement statement = connection.prepareStatement(sql)) {

statement.setInt(1, userId);

return statement.executeUpdate() > 0;

} catch (SQLException e) {

e.printStackTrace();

return false;

}

}

*@Override*

public boolean deleteProject(int projectId) {

String sql = "DELETE FROM Project WHERE id = ?";

try (PreparedStatement statement = connection.prepareStatement(sql)) {

statement.setInt(1, projectId);

return statement.executeUpdate() > 0;

} catch (SQLException e) {

e.printStackTrace();

return false;

}

}

*@Override*

public List<Task> getAllTasks(int empId, int projectId) {

List<Task> tasks = new ArrayList<>();

String sql = "SELECT \* FROM Task WHERE employee\_id = ? AND project\_id = ?";

try (PreparedStatement statement = connection.prepareStatement(sql)) {

statement.setInt(1, empId);

statement.setInt(2, projectId);

ResultSet rs = statement.executeQuery();

while (rs.next()) {

Task task = new Task(

rs.getInt("task\_id"),

rs.getString("task\_name"),

rs.getInt("project\_id"),

rs.getInt("employee\_id"),

rs.getString("status")

);

tasks.add(task);

}

} catch (SQLException e) {

e.printStackTrace();

}

return tasks;

}

*@Override*

public boolean deleteTask(int taskId) {

String query = "DELETE FROM Task WHERE task\_id = ?";

try (PreparedStatement stmt = connection.prepareStatement(query)) {

stmt.setInt(1, taskId);

return stmt.executeUpdate() > 0;

} catch (SQLException e) {

e.printStackTrace();

return false;

}

}

*@Override*

public List<Project> searchProjectsByEmployeeId(int empId) {

List<Project> projects = new ArrayList<>();

String sql = "SELECT p.\* FROM Project p JOIN Employee e ON p.id = e.project\_id WHERE e.id = ?";

try (PreparedStatement statement = connection.prepareStatement(sql)) {

statement.setInt(1, empId);

ResultSet rs = statement.executeQuery();

while (rs.next()) {

Project project = new Project(

rs.getInt("id"),

rs.getString("project\_name"),

rs.getString("description"),

rs.getString("start\_date"),

rs.getString("status")

);

projects.add(project);

}

} catch (SQLException e) {

e.printStackTrace();

}

if (projects.isEmpty()) {

throw new EmployeeNotFoundException("Employee not found with id: " + empId);

}

return projects;

}

*@Override*

public List<Task> searchTasksByEmployeeId(int empId) {

List<Task> tasks = new ArrayList<>();

String sql = "SELECT \* FROM Task WHERE employee\_id = ?";

try (PreparedStatement statement = connection.prepareStatement(sql)) {

statement.setInt(1, empId);

ResultSet rs = statement.executeQuery();

while (rs.next()) {

Task task = new Task(

rs.getInt("task\_id"),

rs.getString("task\_name"),

rs.getInt("project\_id"),

rs.getInt("employee\_id"),

rs.getString("status")

);

tasks.add(task);

}

} catch (SQLException e) {

e.printStackTrace();

}

if (tasks.isEmpty()) {

throw new EmployeeNotFoundException("Employee not found with id: " + empId);

}

return tasks;

}

*@Override*

public List<Task> searchTasksByProjectId(int projectId) {

List<Task> tasks = new ArrayList<>();

String sql = "SELECT \* FROM Task WHERE project\_id = ?";

try (PreparedStatement statement = connection.prepareStatement(sql)) {

statement.setInt(1, projectId);

ResultSet rs = statement.executeQuery();

while (rs.next()) {

Task task = new Task(

rs.getInt("task\_id"),

rs.getString("task\_name"),

rs.getInt("project\_id"),

rs.getInt("employee\_id"),

rs.getString("status")

);

tasks.add(task);

}

} catch (SQLException e) {

e.printStackTrace();

}

if (tasks.isEmpty()) {

throw new ProjectNotFoundException("Project not found with id: " + projectId);

}

return tasks;

}

}

package entity;

public class Employee {

private int id;

private String name;

private String designation;

private String gender;

private double salary;

private Integer projectId;

public Employee(String name, String designation, String gender, double salary, Integer projectId) {

this.name = name;

this.designation = designation;

this.gender = gender;

this.salary = salary;

this.projectId = projectId;

}

public Employee(int id, String name, String designation, String gender, double salary, Integer projectId) {

this.id = id;

this.name = name;

this.designation = designation;

this.gender = gender;

this.salary = salary;

this.projectId = projectId;

}

// Getters and Setters

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getDesignation() {

return designation;

}

public void setDesignation(String designation) {

this.designation = designation;

}

public String getGender() {

return gender;

}

public void setGender(String gender) {

this.gender = gender;

}

public double getSalary() {

return salary;

}

public void setSalary(double salary) {

this.salary = salary;

}

public Integer getProjectId() {

return projectId;

}

public void setProjectId(Integer projectId) {

this.projectId = projectId;

}

}

package entity;

public class Project {

private int id;

private String projectName;

private String description;

private String startDate;

private String status;

public Project(String projectName, String description, String startDate, String status) {

this.projectName = projectName;

this.description = description;

this.startDate = startDate;

this.status = status;

}

public Project(int id, String projectName, String description, String startDate, String status) {

this.id = id;

this.projectName = projectName;

this.description = description;

this.startDate = startDate;

this.status = status;

}

// Getters and Setters

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getProjectName() {

return projectName;

}

public void setProjectName(String projectName) {

this.projectName = projectName;

}

public String getDescription() {

return description;

}

public void setDescription(String description) {

this.description = description;

}

public String getStartDate() {

return startDate;

}

public void setStartDate(String startDate) {

this.startDate = startDate;

}

public String getStatus() {

return status;

}

public void setStatus(String status) {

this.status = status;

}

}

package entity;

public class Task {

private int taskId;

private String taskName;

private int projectId;

private int employeeId;

private String status;

public Task(String taskName, int projectId, int employeeId, String status) {

this.taskName = taskName;

this.projectId = projectId;

this.employeeId = employeeId;

this.status = status;

}

public Task(int taskId, String taskName, int projectId, int employeeId, String status) {

this.taskId = taskId;

this.taskName = taskName;

this.projectId = projectId;

this.employeeId = employeeId;

this.status = status;

}

// Getters and Setters

public int getTaskId() {

return taskId;

}

public void setTaskId(int taskId) {

this.taskId = taskId;

}

public String getTaskName() {

return taskName;

}

public void setTaskName(String taskName) {

this.taskName = taskName;

}

public int getProjectId() {

return projectId;

}

public void setProjectId(int projectId) {

this.projectId = projectId;

}

public int getEmployeeId() {

return employeeId;

}

public void setEmployeeId(int employeeId) {

this.employeeId = employeeId;

}

public String getStatus() {

return status;

}

public void setStatus(String status) {

this.status = status;

}

}

package myexceptions;

public class EmployeeNotFoundException extends RuntimeException {

public EmployeeNotFoundException(String message) {

super(message);

}

}

package myexceptions;

public class ProjectNotFoundException extends RuntimeException {

public ProjectNotFoundException(String message) {

super(message);

}

}

package test;

import static org.junit.jupiter.api.Assertions.\*;

import org.junit.jupiter.api.Test;

import dao.ProjectRepositoryImpl;

import entity.Employee;

public class EmployeeTest {

@Test

public void testCreateEmployeeSuccess() {

ProjectRepositoryImpl repo = new ProjectRepositoryImpl();

Employee emp = new Employee("Alice", "Developer", "Female", 60000, null);

boolean result = repo.createEmployee(emp);

assertTrue(result, "Employee should be created successfully");

}

}

package test;

import static org.junit.jupiter.api.Assertions.\*;

import dao.ProjectRepositoryImpl;

import myexceptions.EmployeeNotFoundException;

import myexceptions.ProjectNotFoundException;

import org.junit.jupiter.api.Test;

public class ExceptionTest {

@Test

public void testEmployeeNotFoundExceptionForProjects() {

ProjectRepositoryImpl repo = new ProjectRepositoryImpl();

Exception exception = assertThrows(EmployeeNotFoundException.class, () -> {

repo.searchProjectsByEmployeeId(999); // assuming 999 is invalid

});

assertEquals("Employee not found with id: 999", exception.getMessage());

}

@Test

public void testEmployeeNotFoundExceptionForTasks() {

ProjectRepositoryImpl repo = new ProjectRepositoryImpl();

Exception exception = assertThrows(EmployeeNotFoundException.class, () -> {

repo.searchTasksByEmployeeId(999); // assuming 999 is invalid

});

assertEquals("Employee not found with id: 999", exception.getMessage());

}

@Test

public void testProjectNotFoundException() {

ProjectRepositoryImpl repo = new ProjectRepositoryImpl();

Exception exception = assertThrows(ProjectNotFoundException.class, () -> {

repo.searchTasksByProjectId(999); // assuming 999 is invalid

});

assertEquals("Project not found with id: 999", exception.getMessage());

}

}

package test;

import static org.junit.jupiter.api.Assertions.\*;

import dao.ProjectRepositoryImpl;

import entity.Project;

import entity.Task;

import org.junit.jupiter.api.Test;

import java.util.List;

public class SearchTest {

@Test

public void testSearchProjectByEmployeeId() {

ProjectRepositoryImpl repo = new ProjectRepositoryImpl();

List<Project> projects = repo.searchProjectsByEmployeeId(1); // assuming 1 is a valid empId

assertNotNull(projects, "Returned list should not be null");

assertFalse(projects.isEmpty(), "Projects should be returned for valid employee ID");

}

@Test

public void testSearchTaskByEmployeeId() {

ProjectRepositoryImpl repo = new ProjectRepositoryImpl();

List<Task> tasks = repo.searchTasksByEmployeeId(1); // assuming 1 is a valid empId

assertNotNull(tasks, "Returned list should not be null");

assertFalse(tasks.isEmpty(), "Tasks should be returned for valid employee ID");

}

}

package test;

import static org.junit.jupiter.api.Assertions.\*;

import org.junit.jupiter.api.Test;

import dao.ProjectRepositoryImpl;

import entity.Task;

public class TaskTest {

@Test

public void testCreateTaskSuccess() {

ProjectRepositoryImpl repo = new ProjectRepositoryImpl();

Task task = new Task("Design UI", 1, 1, "ASSIGNED");

boolean result = repo.createTask(task);

assertTrue(result, "Task should be created successfully");

}

}

package util;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

import java.util.Properties;

public class DBConnUtil {

private static Connection connection = null;

public static Connection getConnection(String propertyFileName) {

if (connection == null) {

try {

Properties props = DBPropertyUtil.getConnectionProperties(propertyFileName);

String url = props.getProperty("db.url");

String user = props.getProperty("db.username");

String password = props.getProperty("db.password");

connection = DriverManager.getConnection(url, user, password);

System.out.println("Database connection established.");

} catch (SQLException e) {

e.printStackTrace();

}

}

return connection;

}

public static void closeConnection() {

if (connection != null) {

try {

connection.close();

System.out.println("Database connection closed.");

} catch (SQLException e) {

e.printStackTrace();

}

}

}

}

package util;

import java.io.FileInputStream;

import java.io.IOException;

import java.io.InputStream;

import java.util.Properties;

public class DBPropertyUtil {

public static Properties getConnectionProperties(String filename) {

Properties props = new Properties();

try (InputStream input = new FileInputStream(filename)) {

props.load(input);

} catch (IOException e) {

e.printStackTrace();

}

return props;

}

}