Case Study Crime Analysis and Reporting System (C.A.R.S.)

Instructions

- Project submissions should **be a single zip file containing source code and an additional document with output screenshot**
- Each section builds upon the previous one, and by the end, you will have a comprehensive **Crime Analysis and Reporting System** implemented with a strong focus on **SQL schema design**, **control flow statements, loops, arrays, collections, exception handling** and database interaction using **JDBC**.
- Remember to provide clear instructions, sample code, and any necessary resources or databases for each assignment.
- Follow **object-oriented principles** throughout the project. Use classes and objects to model real-world entities, **encapsulate data and behavior**, and **ensure code reusability**.
- Throw **user defined exceptions** from corresponding methods and handle in the main method**.**

1: Problem Statement:

Insurance Management System using Core Java with JDBC

Introduction: The primary objective of this project is to develop a comprehensive **Crime Analysis and Reporting System (CARS)** that addresses the above-mentioned challenges and provides law enforcement agencies with a robust, user-friendly, and secure platform for crime data management and reporting

1. Schema design:

Entities:

1. **Incidents:**
    - IncidentID (Primary Key)

- IncidentType (e.g., Robbery, Homicide, Theft)
- IncidentDate
- Location (Geospatial Data: Latitude and Longitude)
- Description
- Status (e.g., Open, Closed, Under Investigation)
- VictimID (Foreign Key, linking to Victims)
- SuspectId(Foreign Key, Linking to Suspect)

2. **Victims:**
    - VictimID (Primary Key)
    - FirstName
    - LastName
    - DateOfBirth
    - Gender
    - Contact Information (e.g., Address, Phone Number)
3. **Suspects:**
    - SuspectID (Primary Key)
    - FirstName
    - LastName
    - DateOfBirth
    - Gender
    - Contact Information

4. **Law Enforcement Agencies:**

- AgencyID (Primary Key)
- AgencyName
- Jurisdiction
- Contact Information
- Officer(s) (Link to Officers within the agency)

5. **Officers:**
    - OfficerID (Primary Key)
    - FirstName
    - LastName
    - BadgeNumber
    - Rank
    - Contact Information
    - AgencyID (Foreign Key, linking to Law Enforcement Agencies)
6. **Evidence:**
    - EvidenceID (Primary Key)
    - Description
    - Location Found
    - IncidentID (Foreign Key, linking to Incidents)
7. **Reports:**
    - ReportID (Primary Key)
    - IncidentID (Foreign Key, linking to Incidents)
    - ReportingOfficer (Foreign Key, linking to Officers)
    - ReportDate

- ReportDetails
- Status (e.g., Draft, Finalized)

Relationships:

- An Incident can have multiple Victims and Suspects.
- An Incident is associated with one Law Enforcement Agency.
- An Officer works for a single Law Enforcement Agency.
- Evidence can be linked to an Incident.
- Reports are generated for Incidents by ReportingOfficers.

.2: Java Model classes

- The following **package structure** to be followed in the **demo application.**

- **com.hexaware.entity**
  - Create entity classes in this package. All entity class should not have any business logic.
- **com.hexaware.dao**
  - Create Service Provider interface to showcase functionalities
  - Create the implementation class for the above interface with db interaction.
- **com.hexaware.exception**

- Create user defined exceptions in this package and handle exceptions whenever needed.

- **com.hexaware.util**

  - Create a **DBPropertyUtil** class with a static function which Takes property file name as parameter and returns connection string.
  - Create a **DBConnUtil** class which holds **static method** which takes connection string as parameter file and returns **connection object(Use method defined in DBPropertyUtil class to get the connection String)**.Top of Form

Scope : Entity classes/Models/POJO, Abstraction/Encapsulation

Create the model**/entity classes corresponding to the schema** within package **com.hexaware.entities** with variables declared private, constructors(default and parametrized,getters,setters and toString())

6: Service Provider Interface

Keep the interfaces and implementation classes in package com.hexaware.dao

Create ICrimeAnalysisService Interface

public interface **ICrimeAnalysisService** {

// **Create a new incident**
Incident createIncident(**Incident** obj);

// **Update the status of an incident**
void updateIncidentStatus(int incidentId, Status newStatus);

```
    // Get a list of incidents within a date range
    List<Incident> getIncidentsInDateRange(Date startDate, Date endDate);

    // Search for incidents based on various criteria
    List<Incident> searchIncidents(IncidentType criteria);

    // Generate incident reports
    Report generateIncidentReport(Incident incident);

    // Create a new case and associate it with incidents
    Case createCase(String caseDescription, List<Incident> relatedIncidents);

    // Get details of a specific case
    Case getCaseDetails(int caseId);

    // Update case details
    void updateCaseDetails(Case caseToUpdate);

    // Get a list of all cases
    List<Case> getAllCases();
}
```

7: JDBC and Database Interaction

Scope: Connection class, Resultset, execute(),executeQuery(),SQL exception, datatype compatibility

Task: Connect your application to the SQL database **:**

1. Write code to establish a connection to your SQL database.

   Create a utility class **DBConnection** in a package **com.hexaware.util** with a static variable **connection** of Type **Connection** and a static method **getConnection()** which returns connection.

   Connection properties supplied in the connection string should be read from a property file.

Create a utility class **PropertyUtil** which contains a static method named **getPropertyString()** which reads a property fie containing connection details like hostname, dbname, username, password, port number and returns a connection string.

7: Service implementation

(Scope: Interfaces, Implementation classes, Object Arrays, Collections)

1. Create a Service class **CrimeAnalysisServiceImpl** in **com.hexaware.dao** with a static variable named connection of type **Connection** which can be assigned in the constructor by invoking the **getConnection()** method in **DBConnection** class

2. Provide implementation for all the methods in the interface by using jdbc.

8: Exception Handling

(Scope: User Defined Exception/Checked /Unchecked Exception/Exception handling using try..catch finally,thow & throws keyword usage)

Create the exceptions in package **com.hexaware.myexceptions**

Define the following custom exceptions and throw them in methods whenever needed. Handle all the exceptions in main method,

1. **IncidentNumberNotFoundException** :throw this exception when user enters an invalid patient number which doesn't exist in db

9. Main Method

Create class named MainModule with main method in com.hexware.mainmod.

Trigger all the methods in service implementation class.

```
-----------------------
Crime Analysis and Reporting System
-----------------------

Main Menu:
1. View Incidents
2. Add Incident
3. Search Incidents
4. Generate Reports
5. Manage Cases
6. Exit

Please enter your choice:
```

1. **View Incidents**: Allows users to view a list of incidents with basic information (e.g., incident ID, type, date, status).
2. **Add Incident**: Permits users to input details of a new incident, including type, date, location, description, and status.
3. **Search Incidents**: Enables users to search for specific incidents based on criteria such as incident type, date range, or location.
4. **Generate Reports**: Provides options to generate various reports, such as incident summaries, statistics, or case-related reports.
5. **Manage Cases**: Allows for the creation and management of case records, which group related incidents together.
6. **Exit**: Exits the system.

10. Unit Testing

Creating JUnit test cases for a **Crime Analysis and Reporting System** is essential to ensure the correctness and reliability of your system. Below are some example questions to guide the creation of JUnit test cases for various components of the system:

1. **Incident Creation:**
    - Does the createIncident method correctly create an incident with the provided attributes?
    - Are the attributes of the created incident accurate?
2. **Incident Status Update:**
    - Does the updateIncidentStatus method effectively update the status of an incident?
    - Does it handle invalid status updates appropriately?
3. **Incident Retrieval:**
    - Can the getIncidentsInDateRange method accurately retrieve incidents within a specified date range?
    - Does it return the expected results, and is it inclusive of the date range bounds?
4. **Incident Search:**
    - Does the searchIncidents method return the correct incidents based on various search criteria (e.g., type, location)?
    - Does it handle cases where no matching incidents are found?
5. **Incident Reporting:**
    - Does the generateIncidentReport method generate accurate reports for specific incidents?
    - Are the report contents consistent with the incident's information?
6. **Error Handling:**
    - Are exceptions and error conditions handled appropriately in the service methods?

- Are the correct exceptions thrown in response to invalid or exceptional conditions?

These questions should help you formulate JUnit test cases that cover a wide range of scenarios and ensure the robustness and reliability of your Crime Analysis and Reporting System. You can create JUnit test classes and methods to address each of these questions and more, depending on the specific requirements and features of your system.