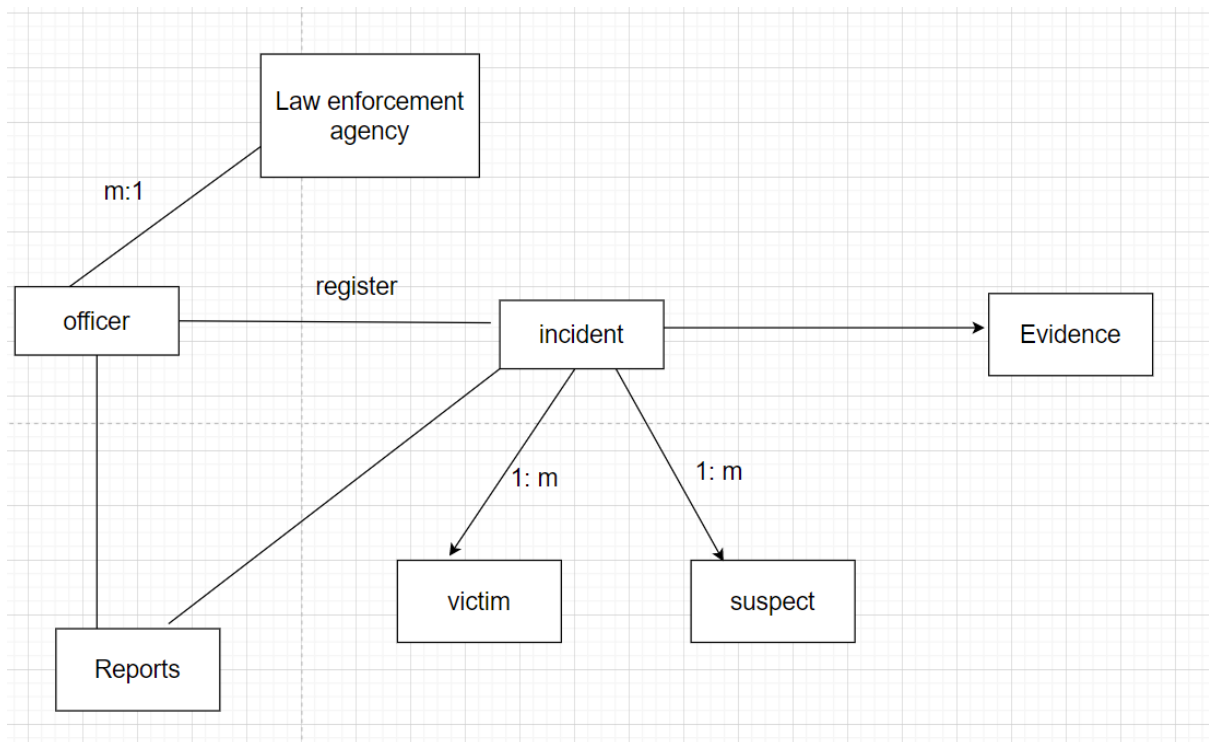


## Crime Analysis and Reporting ( C A R S )



### Instructions

- Project submissions should be done through the participants' Github repository and the link should be shared with trainers and Hexavarsity.
- 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, control flow statements, loops, arrays, collections, exception handling, database interaction and Unit Testing.
- 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 handled.

- The following Directory structure is to be followed in the application.
  - **entity**
    - Create entity classes in this package. All entity class should not have any business logic.
  - **dao**
    - Create Service Provider interface to showcase functionalities.
    - Create the implementation class for the above interface with db interaction.
  - **exception**
    - Create user defined exceptions in this package and handle exceptions whenever needed.
  - **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).
  - **main**
    - Create a class MainModule and demonstrate the functionalities in a menu driven application.

#### **Key Functionalities:**

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.

```

// Create a new incident
createIncident();
    parameters- Incident object
    return type Boolean

// Update the status of an incident
updateIncidentStatus();
    parameters- Status object,incidentid
    return type Boolean
// Get a list of incidents within a date range
getIncidentsInDateRange();
    parameters- startDate, endDate
    return type Collection of Incident objects

// Search for incidents based on various criteria
searchIncidents(IncidentType criteria);
    parameters- IncidentType object
    return type Collection of Incident objects
// Generate incident reports
generateIncidentReport();
    parameters- Incident object
    return type Report object
// Create a new case and associate it with incidents
createCase();
    parameters- caseDescription string, collection of Incident Objects
    return type Case object
// Get details of a specific case
Case getCaseDetails(int caseId);
    parameters- caseDescription string, collection of Incident Objects
    return type Case object

// Update case details
updateCaseDetails();
    parameters- Case object
    return type boolean
// Get a list of all cases
List<Case> getAllCases();
    parameters- None
    return type Collection of cases

```

## 8: Exception Handling

Create the exceptions in package **c.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

## 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?