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

Project Title: Crime Analysis and Reporting System (C.A.R.S.)
Submitted By: C.S.Deepta
Date: 07/04/2025
Trainer: Mrs.Karthika Madan
Organization: Hexaware

ABSTRACT

The Crime Analysis and Reporting System is a Python-based application designed to streamline the management, analysis, and reporting of crime-related data. This system facilitates the efficient recording of crime incidents, tracking their statuses, generating reports, and associating incidents with criminal cases. Developed using an object oriented approach, the system ensures modularity and maintainability. It integrates with a MySQL database to store and retrieve data securely. Core features include the creation and update of incidents, case management, dynamic reporting, and robust exception handling to ensure system stability. The project includes unit testing to validate the correctness of service functionalities. The system's architecture is layered, comprising entity models, data access objects, utility classes for database connection, and custom exception handling. Designed for use by law enforcement agencies or investigative departments, this project demonstrates the practical application of programming principles and database management to solve real-world problems.

Keywords - Crime Analysis, Crime Reporting, Incident Management, Case Management, MySQL Database, Object-Oriented Programming, Exception Handling, Unit Testing, Service Interface, Database Connectivity, PyCharm, SQL Integration, Criminal Record System.

TABLE OF CONTENTS

PURPOSE OF THE PROJECT

The **Crime Analysis and Reporting System (C.A.R.S.)** has been developed to serve as a robust, scalable, and user-friendly solution for managing crime-related data. The main purpose of this project is to empower law enforcement agencies with a centralized platform for efficiently recording, analyzing, and reporting criminal activities.

Key objectives include:

- **SQL and Database Design** for structured data storage and retrieval.
- **Control Flow, Loops, and Exception Handling** for robust application behavior.
- **Object-Oriented Programming** to represent real-world entities like Officers, Incidents, Victims, etc.
- **User-Defined Exceptions** to ensure reliable error reporting and handling.
- **Menu-Driven Console Interface** for easy navigation and use.
- **Unit Testing** to validate core functionalities and ensure system reliability.

Overall, the project is designed to create a comprehensive and reliable **Crime Analysis and Reporting System** using Python, SQL, and Object-Oriented Programming principles to facilitate the managing of the crime related data.

SCOPE OF THE PROJECT

Overview:

The Crime Analysis and Reporting System (C.A.R.S.) aims to streamline crime data management using Python. It covers functionalities such as incident creation, status updates, and report generation. The system handles entities like victims, suspects, officers, and law enforcement agencies. It ensures secure database connectivity and efficient information retrieval. Exception handling and unit testing are implemented to ensure system robustness. The project supports future scalability for analytical and reporting enhancements.

MODULES AND STRUCTURES

The system consists of multiple modules, each focusing on specific operations such as managing incidents, victims, suspects, officers, reports, and evidence. The main components are described below.

1. Database Design:

I. Entities:

- **Incidents**: Stores data about crime events including type, date, description, location, status.
- **Victims**: Stores victim details such as name, gender, DOB, and contact.
- **Suspects**: Contains suspect information like name, DOB, and contact.
- **Officers**: Officers who investigate incidents; includes ID, rank, and badge number.
- **Agencies**: Law enforcement agencies and their jurisdiction/contact details.
- **Evidence**: Items linked to an incident, with location and description.
- **Reports**: Reports written by officers for specific incidents.

II. Relationships:

- One **Incident** can involve multiple **Victims** and **Suspects**.
- Each **Incident** is linked to one **Agency**.
- An **Officer** belongs to one **Agency**.
- Multiple **Evidence** records can be linked to one **Incident**.
- **Reports** are generated for **Incidents** by specific **Officers**.

III SQL Tables & Schema:

- Each entity corresponds to a table with proper primary and foreign keys.
- Relations are handled via one-to-many and many-to-one connections.
- Normalized structure for efficient querying and scalability.

2. Python Program Structure:

I. User Authentication :

- Login system for officers/admin using secure credential checks.

II. Incident Management:

- Create, update, and retrieve incident details.
- Filter incidents based on date, type, or status.
- Link incidents with suspects, victims, and agencies.

III. Report and Evidence Management:

- Officers can write and finalize reports for incidents.
- Evidence items can be added and linked to incidents.

IV. Officer & Agency Management:

- Add/update officer details and assign them to agencies.
- Agency details like jurisdiction and contacts are stored.

V. Case & Exception Handling:

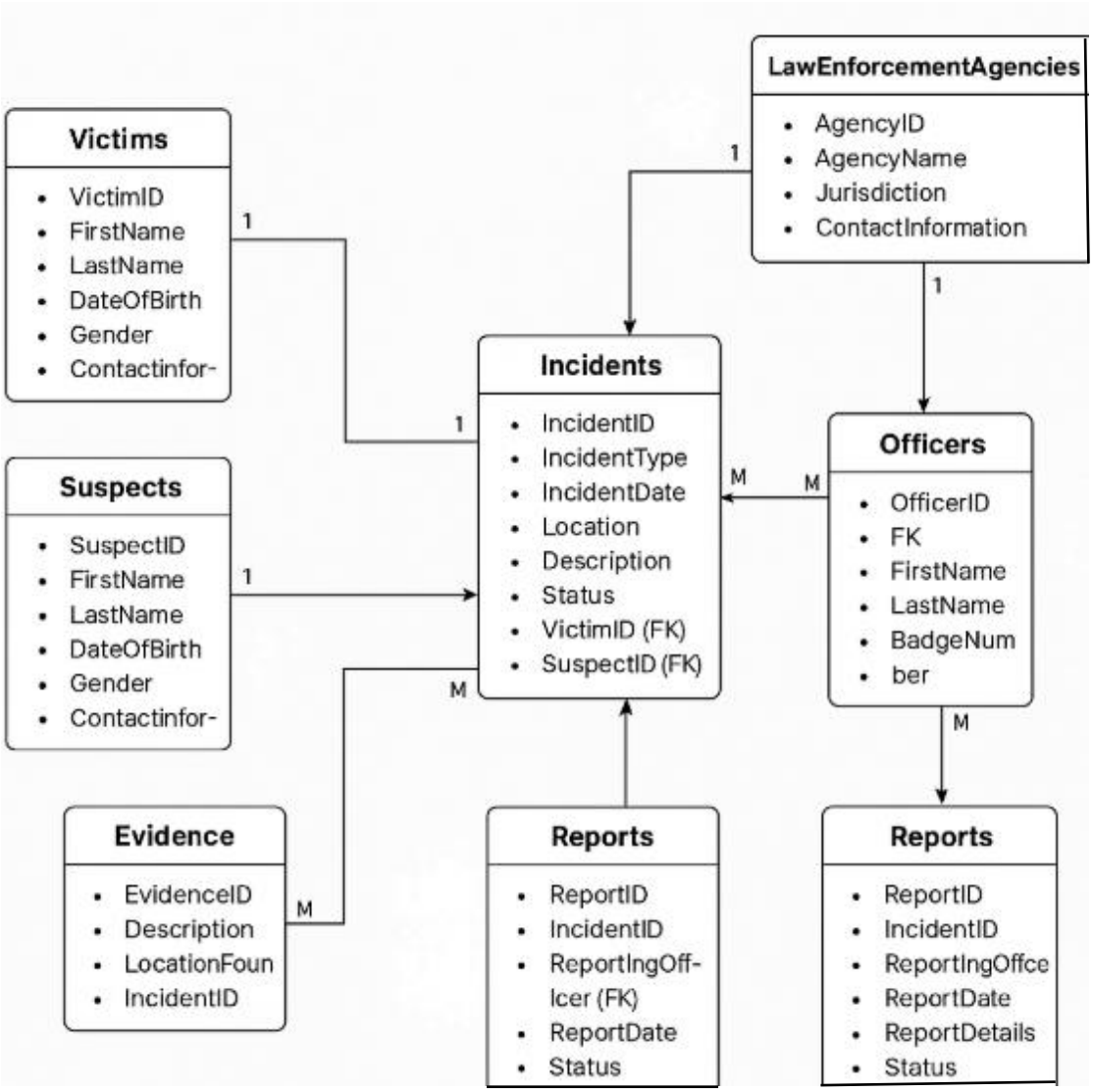
- Create and manage cases linking multiple incidents.
- Handle exceptions like IncidentNumberNotFoundException and InvalidOfficerException.

TECHNOLOGIES USED FOR THE PROJECT

1. **MySQL:** Used as the relational database management system to store and manage crime details, incidents, and tracking information. SQL queries are employed to insert, update, and retrieve data from the database.
2. **PyCharm:** The Integrated Development Environment (IDE) used for Python development. It is utilized for writing, debugging, and testing the Python code for the Crime Analysis and Reporting System
3. **GitHub:** A platform for version control and collaboration. GitHub is used to manage the project's source code, track changes, and collaborate with team members, ensuring efficient version control throughout the development process.

ER DIAGRAM

An ER (Entity-Relationship) Diagram is a visual representation of the entities within a system and the relationships between them. It helps in designing and understanding the database structure by mapping out tables, attributes, and how they are interconnected. The **ER Diagram** for the Crime Analysis and Reporting System illustrates the key entities involved in the system and their interrelationships. It is designed to efficiently manage and analyze crime-related data while maintaining proper referential integrity.



Entities and Their Attributes:

- Victims
  → Attributes:
  • VictimID (Primary Key)
  • FirstName
  • LastName
  • DateOfBirth
  • Gender
  • ContactInformation

- Suspects
  → Attributes:
  • SuspectID (Primary Key)
  • FirstName
  • LastName
  • DateOfBirth

• Gender
• ContactInformation

- LawEnforcementAgencies
  → Attributes:
  • AgencyID (Primary Key)
  • AgencyName
  • Jurisdiction
  • ContactInformation

- Officers
  → Attributes:
  • OfficerID (Primary Key)
  • FirstName
  • LastName
  • BadgeNumber (Unique)
  • Rank
  • ContactInformation
  • AgencyID (Foreign Key → LawEnforcementAgencies)

- Incidents
  → Attributes:
  • IncidentID (Primary Key)
  • IncidentType
  • IncidentDate
  • Location
  • Description
  • Status
  • VictimID (Foreign Key → Victims)
  • SuspectID (Foreign Key → Suspects)

- Evidence
  → Attributes:
  • EvidenceID (Primary Key)
  • Description
  • LocationFound
  • IncidentID (Foreign Key → Incidents)

- Reports
  → Attributes:
  • ReportID (Primary Key)
  • IncidentID (Foreign Key → Incidents)
  • ReportingOfficer (Foreign Key → Officers)
  • ReportDate
  • ReportDetails
  • Status

Relationships:

- A **LawEnforcementAgency** can have **many Officers** (1:M).

- Each **Officer** belongs to **one LawEnforcementAgency** (M:1).

- Each **Incident** can be related to **one Victim** and **one Suspect** (M:1).

- An **Incident** can have **multiple Evidence records** (1:M).

- An **Incident** can be associated with **multiple Reports**, but each **Report** is written for one Incident only (1:M).

- Each **Report** is created by a **single Officer**, but an **Officer** can create **multiple Reports** (1:M).

PART -1 SQL

TASK

1) Database Design for Crime Analysis and Reporting System

Objective:

The goal of this task is to design a relational database schema for the Crime Analysis and Reporting System. The schema should include tables for core entities such as Victims, Suspects, Officers, Incidents and other related entities. We will define the relationships between these tables using foreign keys and populate the tables with sample data to simulate real-world scenarios.

Step 1: Creating Database

To begin, we will create a database called CrimeAnalysis in MySQL. This will serve as the container for all the tables related to the C.A.R.S.

```
CREATE DATABASE CrimeAnalysis;
USE CrimeAnalysis;
```

Step 2 : Creating Table

- **Incidents Table:** The Incidents table stores information about criminal incidents reported in the system. It includes details such as the type, date, location, description, and status of the incident. It also links to the victim and suspect involved in the incident.

  Attributes:

  - **IncidentID:** The primary key, uniquely identifying each incident.
  - **IncidentType:** The type of incident (e.g., Robbery, Homicide, Theft).
  - **IncidentDate:** The date when the incident occurred.
  - **Location:** The geographical location of the incident
  - **Description:** A detailed description of the incident.

- **Status:** The current status of the incident (e.g., Open, Closed, Under Investigation).
- **VictimID:** Foreign key linking to the Victim table.
- **SuspectID:** Foreign key linking to the Suspect table.

```
CREATE TABLE Incidents (
   IncidentID INT PRIMARY KEY AUTO_INCREMENT,
   IncidentType VARCHAR(100),
   IncidentDate DATE,
   Location VARCHAR(255),
   Description TEXT,
   Status VARCHAR(50),
   VictimID INT,
   SuspectID INT,
   FOREIGN KEY (VictimID) REFERENCES Victims(VictimID),
   FOREIGN KEY (SuspectID) REFERENCES Suspects(SuspectID)
);
```

- **Victims Table:** The Victims table contains information about individuals who are victims in various incidents.

  Attributes:

  - **VictimID:** The primary key, uniquely identifying each victim.
  - **FirstName:** Victim's first name.
  - **LastName:** Victim's last name.
  - **DateOfBirth:** Victim's date of birth.
  - **Gender:** Victim's gender.
  - **ContactInformation:** Victim's contact details, including address and phone number.

```
CREATE TABLE Victims (
   VictimID INT PRIMARY KEY AUTO_INCREMENT,
   FirstName VARCHAR(100),
   LastName VARCHAR(100),
   DateOfBirth DATE,
   Gender VARCHAR(10),
   ContactInformation VARCHAR(255)
);
```

- **Suspects Table:** The Suspects table maintains data about individuals suspected of being involved in criminal activities.

  Attributes:

  - **SuspectID:** The primary key, uniquely identifying each suspect.
  - **FirstName:** Suspect's first name.
  - **LastName:** Suspect's last name.
  - **DateOfBirth:** Suspect's date of birth.
  - **Gender:** Suspect's gender.
  - **ContactInformation:** Suspect's contact details including address and phone number.

```
CREATE TABLE Suspects (
   SuspectID INT PRIMARY KEY AUTO_INCREMENT,
   FirstName VARCHAR(100),
   LastName VARCHAR(100),
   DateOfBirth DATE,
   Gender VARCHAR(10),
   ContactInformation VARCHAR(255)
);
```

- **LawEnforcementAgencies Table:** This table stores data about the law enforcement agencies involved in investigating incidents.

  Attributes:

  - **AgencyID:** The primary key, uniquely identifying each law enforcement agency.
  - **AgencyName:** Name of the agency.
  - **Jurisdiction:** The geographical area covered by the agency.
  - **ContactInformation:** Contact details of the agency.
  - **OfficerID:** Links to officers associated with this agency.

```
CREATE TABLE LawEnforcementAgencies (
   AgencyID INT PRIMARY KEY AUTO_INCREMENT,
   AgencyName VARCHAR(100),
   Jurisdiction VARCHAR(100),
   ContactInformation VARCHAR(255)
);
```

- **Officers Table:** The Officers table stores information about officers working in law enforcement agencies.

  Attributes:

  - **OfficerID:** The primary key, uniquely identifying each officer.
  - **FirstName:** Officer's first name.
  - **LastName:** Officer's last name.
  - **BadgeNumber:** Unique badge number of the officer.
  - **Rank:** Rank of the officer (e.g., Inspector, Constable).
  - **ContactInformation:** Officer's contact details.
  - **AgencyID:** Foreign key linking to the Law Enforcement Agency table.

```
CREATE TABLE Officers (
   OfficerID INT PRIMARY KEY AUTO_INCREMENT,
   FirstName VARCHAR(100),
   LastName VARCHAR(100),
   BadgeNumber VARCHAR(100) UNIQUE,
   `Rank` VARCHAR(50),  -- Used backticks to avoid keyword conflict
   ContactInformation VARCHAR(255),
   AgencyID INT,
   FOREIGN KEY (AgencyID) REFERENCES LawEnforcementAgencies(AgencyID)
);
```

- **Evidence Table:** This table records the evidence collected from incident scenes.

Attributes:

- **EvidenceID:** The primary key, uniquely identifying each piece of evidence.
- **Description:** Description of the evidence.
- **LocationFound:** The place where the evidence was found.
- **IncidentID:** Foreign key linking to the associated incident.

```
CREATE TABLE Evidence (
   EvidenceID INT PRIMARY KEY AUTO_INCREMENT,
   Description TEXT,
   LocationFound VARCHAR(255),
   IncidentID INT,
   FOREIGN KEY (IncidentID) REFERENCES Incidents(IncidentID)
);
```

- **Report Table:** The Report table documents reports written for incidents by officers.

Attributes:

- **ReportID:** The primary key, uniquely identifying each report.
- **IncidentID:** Foreign key linking to the Incident table.
- **ReportingOfficer:** Foreign key linking to the Officer table.
- **ReportDate:** The date on which the report was filed.
- **ReportDetails:** Detailed content of the report.
- **Status:** Status of the report.

```
CREATE TABLE Reports (
   ReportID INT PRIMARY KEY AUTO_INCREMENT,
   IncidentID INT,
   ReportingOfficer INT,
   ReportDate DATE,
   ReportDetails TEXT,
   Status VARCHAR(50),
   FOREIGN KEY (IncidentID) REFERENCES Incidents(IncidentID),
   FOREIGN KEY (ReportingOfficer) REFERENCES Officers(OfficerID)
);
```

Step 3 : Inserting values into the table

Inserting values into the tables involves adding data to each table to populate the database with real-world information. This is done using the INSERT INTO SQL command, specifying the table name and the corresponding values for each column. It is important to ensure that the data types and constraints (such as primary and foreign keys) match the table definitions. Proper insertion of values is crucial for maintaining data consistency and integrity across the system. Additionally, sample data helps simulate the actual operations of the Crime Analysis and Reporting System, allowing for effective testing and validation of the system's functionality.

```
INSERT INTO LawEnforcementAgencies (AgencyName, Jurisdiction, ContactInformation)
VALUES
('Mumbai Police', 'Mumbai, India', 'mumbaipolice@gov.in'),
```

('Los Angeles PD', 'Los Angeles, USA', 'lapd@gov.us'),
('Scotland Yard', 'London, UK', 'scotlandyard@gov.uk'),
('New York PD', 'New York, USA', 'nypd@gov.us'),
('Interpol', 'International', 'interpol@gov.int'),
('Delhi Police', 'Delhi, India', 'delhipolice@gov.in'),
('Sydney Police', 'Sydney, Australia', 'sydpolice@gov.au'),
('Toronto Police', 'Toronto, Canada', 'torontopolice@gov.ca'),
('Dubai Police', 'Dubai, UAE', 'dubaipolice@gov.ae'),
('Tokyo Metropolitan Police', 'Tokyo, Japan', 'tokyopolice@gov.jp');

SELECT * FROM LawEnforcementAgencies;

AgencyID	AgencyName	Jurisdiction	ContactInformation
1	Mumbai Police	Mumbai, India	mumbaipolice@gov.in
2	Los Angeles PD	Los Angeles, USA	lapd@gov.us
3	Scotland Yard	London, UK	scotlandyard@gov.uk
4	New York PD	New York, USA	nypd@gov.us
5	Interpol	International	interpol@gov.int
6	Delhi Police	Delhi, India	delhipolice@gov.in
7	Sydney Police	Sydney, Australia	sydpolice@gov.au
8	Toronto Police	Toronto, Canada	torontopolice@gov.ca
9	Dubai Police	Dubai, UAE	dubaipolice@gov.ae
10	Tokyo Metropolitan Police	Tokyo, Japan	tokyopolice@gov.jp
NULL	NULL	NULL	NULL

INSERT INTO Officers (FirstName, LastName, BadgeNumber, `Rank`, ContactInformation, AgencyID) VALUES
('Sheriff', 'Woody', 'TX1001', 'Sergeant', 'woody@lapd.com', 2),
('Buzz', 'Lightyear', 'TX1002', 'Lieutenant', 'buzz@nycpd.com', 4),
('Donald', 'Duck', 'TX1003', 'Detective', 'donald@scotlandyard.com', 3),
('Mickey', 'Mouse', 'TX1004', 'Captain', 'mickey@mumbaipolice.com', 1),
('Tom', 'Cat', 'TX1005', 'Inspector', 'tom@sydpolice.com', 7),
('Jerry', 'Mouse', 'TX1006', 'Constable', 'jerry@delhipolice.com', 6),
('SpongeBob', 'SquarePants', 'TX1007', 'Sergeant', 'spongebob@torontopolice.com', 8),
('Patrick', 'Star', 'TX1008', 'Lieutenant', 'patrick@dubaipolice.com', 9),
('Bugs', 'Bunny', 'TX1009', 'Detective', 'bugs@tokyopolice.com', 10),
('Daffy', 'Duck', 'TX1010', 'Commissioner', 'daffy@interpol.com', 5);

SELECT * FROM Officers;

OfficerID	FirstName	LastName	BadgeNumber	Rank	ContactInformation	AgencyID
1	Sheriff	Woody	TX1001	Sergeant	woody@lapd.com	2
2	Buzz	Lightyear	TX1002	Lieutenant	buzz@nycpd.com	4
3	Donald	Duck	TX1003	Detective	donald@scotlandyard.com	3
4	Mickey	Mouse	TX1004	Captain	mickey@mumbaipolice.com	1
5	Tom	Cat	TX1005	Inspector	tom@sydpolice.com	7
6	Jerry	Mouse	TX1006	Constable	jerry@delhipolice.com	6
7	SpongeBob	SquarePants	TX1007	Sergeant	spongebob@torontopolice.com	8
8	Patrick	Star	TX1008	Lieutenant	patrick@dubaipolice	patrick@dubaipolice.com
9	Bugs	Bunny	TX1009	Detective	bugs@tokyopolice.com	10
10	Daffy	Duck	TX1010	Commissioner	daffy@interpol.com	5

INSERT INTO Victims (FirstName, LastName, DateOfBirth, Gender, ContactInformation)
VALUES
('Shah Rukh', 'Khan', '1965-11-02', 'Male', 'srk@bollywood.com'),
('Salman', 'Khan', '1965-12-27', 'Male', 'salmankhan@bollywood.com'),
('Amitabh', 'Bachchan', '1942-10-11', 'Male', 'amitabh@bollywood.com'),
('Aishwarya', 'Rai', '1973-11-01', 'Female', 'aish@bollywood.com'),
('Priyanka', 'Chopra', '1982-07-18', 'Female', 'priyanka@bollywood.com'),
('Deepika', 'Padukone', '1986-01-05', 'Female', 'deepika@bollywood.com'),
('Ranveer', 'Singh', '1985-07-06', 'Male', 'ranveer@bollywood.com'),
('Hrithik', 'Roshan', '1974-01-10', 'Male', 'hrithik@bollywood.com'),
('Alia', 'Bhatt', '1993-03-15', 'Female', 'alia@bollywood.com'),
('Kareena', 'Kapoor', '1980-09-21', 'Female', 'kareena@bollywood.com');
SELECT * FROM Victims;

VictimID	FirstName	LastName	DateOfBirth	Gender	ContactInformation
1	Shah Rukh	Khan	1965-11-02	Male	srk@bollywood.com
2	Salman	Khan	1965-12-27	Male	salmankhan@bollywood.com
3	Amitabh	Bachchan	1942-10-11	Male	amitabh@bollywood.com
4	Aishwarya	Rai	1973-11-01	Female	aish@bollywood.com
5	Priyanka	Chopra	1982-07-18	Female	priyanka@bollywood.com
6	Deepika	Padukone	1986-01-05	Female	deepika@bollywood.com
7	Ranveer	Singh	1985-07-06	Male	ranveer@bollywood.com
8	Hrithik	Roshan	1974-01-10	Male	hrithik@bollywood.com
9	Alia	Bhatt	1993-03-15	Female	alia@bollywood.com
10	Kareena	Kapoor	1980-09-21	Female	kareena@bollywood.com
NULL	NULL	NULL	NULL	NULL	NULL

INSERT INTO Suspects (FirstName, LastName, DateOfBirth, Gender, ContactInformation)
VALUES
('Johnny', 'Depp', '1963-06-09', 'Male', 'johnny@hollywood.com'),
('Leonardo', 'DiCaprio', '1974-11-11', 'Male', 'leo@hollywood.com'),
('Angelina', 'Jolie', '1975-06-04', 'Female', 'angelina@hollywood.com'),
('Brad', 'Pitt', '1963-12-18', 'Male', 'brad@hollywood.com'),
('Tom', 'Cruise', '1962-07-03', 'Male', 'tom@hollywood.com'),
('Selena', 'Gomez', '1992-07-22', 'Female', 'selena@hollywood.com'),
('Robert', 'Downey Jr.', '1965-04-04', 'Male', 'rdj@hollywood.com'),
('Scarlett', 'Johansson', '1984-11-22', 'Female', 'scarlett@hollywood.com'),
('Chris', 'Evans', '1981-06-13', 'Male', 'chris@hollywood.com'),
('Will', 'Smith', '1968-09-25', 'Male', 'will@hollywood.com');

SELECT * FROM Suspects;

SuspectID	FirstName	LastName	DateOfBirth	Gender	ContactInformation
1	Johnny	Depp	1963-06-09	Male	johnny@hollywood.com
2	Leonardo	DiCaprio	1974-11-11	Male	leo@hollywood.com
3	Angelina	Jolie	1975-06-04	Female	angelina@hollywood.com
4	Brad	Pitt	1963-12-18	Male	brad@hollywood.com
5	Tom	Cruise	1962-07-03	Male	tom@hollywood.com
6	Selena	Gomez	1992-07-22	Female	selena@hollywood.com
7	Robert	Downey Jr.	1965-04-04	Male	rdj@hollywood.com
8	Scarlett	Johansson	1984-11-22	Female	scarlett@hollywood.com
9	Chris	Evans	1981-06-13	Male	chris@hollywood.com
10	Will	Smith	1968-09-25	Male	will@hollywood.com
NULL	NULL	NULL	NULL	NULL	NULL

INSERT INTO Incidents (IncidentType, IncidentDate, Location, Description, Status, VictimID, SuspectID) VALUES
('Robbery', '2024-01-05', 'Mumbai, India', 'Shah Rukh Khan was robbed at a film set.', 'Under Investigation', 1, 1),
('Homicide', '2024-02-10', 'Delhi, India', 'Aamir Khan was found dead in a hotel room.', 'Closed', 2, 2),
('Theft', '2024-03-15', 'Chennai, India', 'Deepika Padukone reported her jewelry stolen.', 'Open', 3, 3),
('Kidnapping', '2024-04-20', 'Hyderabad, India', 'Hrithik Roshan was kidnapped and later found.', 'Closed', 4, 4),
('Assault', '2024-05-25', 'Pune, India', 'Salman Khan was assaulted at an event.', 'Under Investigation', 5, 5),
('Burglary', '2024-06-30', 'Kolkata, India', 'Alia Bhatt's house was broken into.', 'Open', 6, 6),
('Fraud', '2024-07-05', 'Bangalore, India', 'Ranbir Kapoor was scammed in a real estate deal.', 'Under Investigation', 7, 7),
('Murder', '2024-08-10', 'Jaipur, India', 'Kareena Kapoor found dead in her apartment.', 'Closed', 8, 8),
('Cyber Crime', '2024-09-15', 'Lucknow, India', 'Ranveer Singh's social media was hacked.', 'Open', 9, 9),
('Drug Possession', '2024-10-20', 'Ahmedabad, India', 'Ajay Devgn caught with illegal substances.', 'Under Investigation', 10, 10)
('Robbery','2024-10-05','hyderabad','the bank was robbered','open',5,6);

SELECT * FROM Incidents;

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

IncidentID	IncidentType	IncidentDate	Location	Description	Status	VictimID	SuspectID
1	Robbery	2024-01-05	Mumbai, India	Shah Rukh Khan was robbed at a film set.	Under Investigation	1	1
2	Homicide	2025-02-10	Delhi, India	Aamir Khan was found dead in a hotel room.	Closed	2	2
3	Theft	2024-03-15	Chennai, India	Deepika Padukone reported her jewelry stolen.	Open	3	3
4	Kidnapping	2023-04-20	Hyderabad, India	Hrithik Roshan was kidnapped and later found.	Closed	4	4
5	Assault	2024-05-25	Pune, India	Salman Khan was assaulted at an event.	Under Investigation	5	5
6	Burglary	2025-03-30	Kolkata, India	Alia Bhatt's house was broken into.	Open	6	6
7	Fraud	2023-07-05	Bangalore, India	Ranbir Kapoor was scammed in a real estate deal.	Under Investigation	7	7
8	Murder	2024-08-10	Jaipur, India	Kareena Kapoor found dead in her apartment.	Closed	8	8
9	Cyber Crime	2023-09-15	Lucknow, India	Ranveer Singh's social media was hacked.	Open	9	9
10	Drug Possession	2025-01-20	Ahmedabad, India	Ajay Devgn caught with illegal substances.	Under Investigation	10	10
11	Fraud	2025-04-13	Srinagar	Scammers pretend to be charity organizations a...	Open	4	6
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

INSERT INTO Evidence (Description, LocationFound, IncidentID) VALUES
('A stolen diamond ring', 'Mumbai Jewelry Store', 1),
('A bloody knife', 'Hotel Room, Delhi', 2),
('A missing laptop', 'Chennai House', 3),
('Fingerprints on a bottle', 'Hyderabad Park', 4),
('CCTV footage of the assault', 'Pune Event Hall', 5),
('A broken window', 'Alia Bhatt's house, Kolkata', 6),
('Fake property documents', 'Real Estate Office, Bangalore', 7),
('A gun found near the scene', 'Jaipur Apartment', 8),
('Hacked social media account', 'Bangalore Office', 9),
('Illegal drugs found in car', 'Ahmedabad Highway', 10);

SELECT * FROM Evidence;

EvidenceID	Description	LocationFound	IncidentID
1	A stolen diamond ring	Mumbai Jewelry Store	1
2	A bloody knife	Hotel Room, Delhi	2
3	A missing laptop	Chennai House	3
4	Fingerprints on a bottle	Hyderabad Park	4
5	CCTV footage of the assault	Pune Event Hall	5
6	A broken window	Alia Bhatt's house, Kolkata	6
7	Fake property documents	Real Estate Office, Bangalore	7
8	A gun found near the scene	Jaipur Apartment	8
9	Hacked social media account	Bangalore Office	9
10	Illegal drugs found in car	Ahmedabad Highway	10
NULL	NULL	NULL	NULL

INSERT INTO Reports (IncidentID, ReportingOfficer, ReportDate, ReportDetails, Status) VALUES
(1, 1, '2024-01-06', 'Initial investigation started. CCTV footage being reviewed.', 'Under Investigation'),
(2, 2, '2024-02-11', 'Autopsy completed. Case closed as confirmed homicide.', 'Closed'),
(3, 3, '2024-03-16', 'Theft reported. No suspects identified yet.', 'Open'),
(4, 4, '2024-04-21', 'Victim found safe. Kidnappers in custody.', 'Closed'),
(5, 5, '2024-05-26', 'Suspect identified from event footage. Arrest pending.', 'Under Investigation'),
(6, 6, '2024-07-01', 'Entry through window confirmed. Forensics underway.', 'Open'),
(7, 7, '2024-07-06', 'Fraudulent documents submitted. Case under review.', 'Under Investigation'),
(8, 8, '2024-08-11', 'Victim's death ruled as murder. Investigation complete.', 'Closed'),
(9, 9, '2024-09-16', 'Hacking traced to overseas server. Cyber team alerted.', 'Open'),
(10, 10, '2024-10-21', 'Drugs found in suspect's vehicle. Lab results awaited.', 'Under Investigation');

SELECT * FROM Reports;

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

ReportID	IncidentID	ReportingOfficer	ReportDate	ReportDetails	Status
1	1	1	2024-01-06	Robbery reported and under investigation.	In Progress
2	2	2	2025-02-11	Homicide case investigated and closed.	Closed
3	3	3	2024-03-16	Theft reported by victim.	Open
4	4	4	2023-04-21	Kidnapping confirmed and resolved.	Closed
5	5	5	2024-05-26	Assault case reported.	Under Investigation
6	6	6	2025-03-31	Burglary investigation in progress.	Open
7	7	7	2023-07-06	Fraud complaint filed.	Under Investigation
8	8	8	2024-08-11	Murder case solved and closed.	Closed
9	9	9	2023-09-16	Cyber crime reported by victim.	Open
10	10	10	2025-01-21	Drug possession under investigation.	Under Investigation
NULL	NULL	NULL	NULL	NULL	NULL

PART -2 CODING

TASK

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

Folder: entity

This package acts as the foundation of the project, where all the core domain classes (also called models or entity classes) reside. These classes are designed to represent the data objects that the system will handle, such as Incidents, Cases, and Status Updates.Each class here corresponds to a real-world object that will be stored in the database. The fields in the class match the attributes of the database table, and each class is built using the concept of encapsulation — meaning the variables are declared private, and access to them is only possible through getters and setters.

- incident.py

  - This class represents a criminal incident reported in the system.
  - It contains all the necessary fields such as incidentId, incidentType, date, location, description, and possibly a status.
  - It has a default constructor (with no arguments) and a parameterized constructor (which initializes all fields).
  - The purpose of this class is to store all the details about a single reported incident.
  - It will be used when adding new incidents, updating them, fetching data, and generating reports.

```
class Incident:
    def __init__(self, incident_id, incident_type, description, date, status):
        self.__incident_id = incident_id
        self.__incident_type = incident_type
        self.__description = description
        self.__date = date
        self.__status = status


    def get_incident_id(self):
        return self.__incident_id

    def get_incident_type(self):
        return self.__incident_type

    def get_description(self):
        return self.__description

    def get_date(self):
        return self.__date

    def get_status(self):
        return self.__status
    def set_status(self, status):
        self.__status = status
```

- case.py

  - Represents a case, which is a collection or grouping of incidents.
  - Attributes include caseId, caseDescription, and a collection (like a list) of associated Incident objects.

- It is important for higher-level tracking where multiple incidents are part of a bigger criminal case.
- This entity links to the incident class and is used when associating multiple incidents together under one case for investigation and analysis.
- Includes constructors and appropriate getter/setter methods to allow manipulation and data access.

```
class Case:
    def __init__(self, case_id, case_description, incidents):
        self.__case_id = case_id
        self.__case_description = case_description
        self.__incidents = incidents


    def get_case_id(self):
        return self.__case_id

    def get_case_description(self):
        return self.__case_description

    def get_incidents(self):
        return self.__incidents


    def set_case_description(self, case_description):
        self.__case_description = case_description

    def add_incident(self, incident):
        self.__incidents.append(incident)
```

- status.py

    - Represents the status of an incident — like "Open", "Under Investigation", "Closed", etc.
    - It has fields such as incidentId, status, updatedBy, and updateDate.
    - This class is crucial when the status of an incident needs to be updated or fetched for reporting.
    - It supports tracking the progress of a case over time.
    - The class includes a default constructor for flexibility and a parameterized one for quick creation of objects with all values set.

```
class Status:
    def __init__(self, status_id, status_name):
        self.__status_id = status_id
        self.__status_name = status_name


    def get_status_id(self):
        return self.__status_id

    def get_status_name(self):
        return self.__status_name


    def set_status_name(self, status_name):
```

self.__status_name = status_name

Importance of Entity Classes :

- They form the **blueprint** for the data your system will use.
- These classes are used by your **service layer** (in dao/) to transfer data to and from the database.
- They help in **data encapsulation**, following the principles of object-oriented programming.

TASK

Service Provider Interface/Abstract class

Keep the interfaces and implementation classes in package dao
Create ICrimeAnalysisService Interface/abstract classs with the following methods

// **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

Folder: dao

This package (dao stands for **Data Access Object**) contains the **core service interface** and its **implementation** class. This is where all your **business logic** lives.

Interface: icrime_analysis_service.py

This is the service interface/abstract class that defines the contract of the Crime Analysis system.

Description of Each Method in crime_analysis_service_impl.py:

- **createIncident(incident: Incident) -> bool**
  - **Purpose**: Add a new incident to the system.
  - **Parameter**: Takes an Incident object (from entity.incident).
  - **Returns**: True if created successfully, otherwise False.
  - **Use Case**: Whenever a user wants to report a new crime.

- **updateIncidentStatus(status: Status, incident_id: int) -> bool**
  - **Purpose**: Update the status of an existing incident.
  - **Parameter**: A Status object and the incident_id to be updated.
  - **Returns**: Boolean indicating success or failure.
  - **Use Case**: Change from "Open" to "Investigating", etc.

- **getIncidentsInDateRange(start_date, end_date) -> Collection[Incident]**
  - **Purpose**: Fetch incidents between two dates.
  - **Parameters**: start_date and end_date.
  - **Returns**: A collection (list) of matching Incident objects.
  - **Use Case**: When generating reports or statistics for a time period.

- **searchIncidents(criteria: str) -> Collection[Incident]**
  - **Purpose**: Search for incidents based on incident type or keyword.
  - **Parameter**: A criteria string or possibly a full IncidentType object.
  - **Returns**: A list of Incident objects matching the criteria.
  - **Use Case**: Search by keyword like “theft”, “assault”, etc.

- **generateIncidentReport(incident: Incident) -> Report**
  - **Purpose**: Generate a detailed report for a given incident.
  - **Parameter**: The Incident object.

- **Returns**: A Report object (could be defined later in the model).
- **Use Case**: For sharing case summaries with stakeholders or departments.

- **createCase(case_description: str, incidents: Collection[Incident]) -> Case**

  - **Purpose**: Create a new case and link it to a group of incidents.
  - **Parameters**: Description string and a collection of Incident objects.
  - **Returns**: A new Case object.
  - **Use Case**: When police link several similar incidents under one case.

- **getCaseDetails(case_id: int) -> Case**

  - **Purpose**: Get full details of a specific case using its ID.
  - **Parameter**: The case_id.
  - **Returns**: A Case object with all its data and linked incidents.
  - **Use Case**: Case review, court processing, or audit.

- **updateCaseDetails(case: Case) -> bool**

  - **Purpose**: Modify/update existing case details.
  - **Parameter**: The Case object to be updated.
  - **Returns**: Boolean value indicating update status.
  - **Use Case**: Add additional notes, update the case description, etc.

- **getAllCases() -> Collection[Case]**

  - **Purpose**: Fetch all criminal cases stored in the system.
  - **Parameters**: None.
  - **Returns**: A list of all Case objects.
  - **Use Case**: Display to admin or analyst for filtering and tracking.

```
from dao.icrime_analysis_service import ICrimeAnalysisService
from util.db_connection import DBConnection

class CrimeAnalysisServiceImpl(ICrimeAnalysisService):
    def __init__(self):
        self.connection = DBConnection.get_connection()

    def create_incident(self, incident):

        try:
            cursor = self.connection.cursor()
            query = "INSERT INTO Incident (incident_id, incident_type, description, date, location,
status) VALUES (%s, %s, %s, %s, %s, %s)"
            values = (incident.incident_id, incident.incident_type, incident.description, incident.date,
incident.location, incident.status)
            cursor.execute(query, values)
            self.connection.commit()
            return True
        except Exception as e:
            print(f"Error creating incident: {e}")
            return False

    def update_incident_status(self, status, incident_id):
```

```
        try:
            cursor = self.connection.cursor()
            query = "UPDATE Incident SET status = %s WHERE incident_id = %s"
            cursor.execute(query, (status, incident_id))
            self.connection.commit()
            return True
        except Exception as e:
            print(f"Error updating incident status: {e}")
            return False

    def get_incidents_in_date_range(self, start_date, end_date):
        try:
            cursor = self.connection.cursor()
            query = "SELECT * FROM Incident WHERE date BETWEEN %s AND %s"
            cursor.execute(query, (start_date, end_date))
            return cursor.fetchall()
        except Exception as e:
            print(f"Error fetching incidents in date range: {e}")
            return []

    def search_incidents(self, incident_type):
        try:
            cursor = self.connection.cursor()
            query = "SELECT * FROM Incident WHERE incident_type = %s"
            cursor.execute(query, (incident_type,))
            return cursor.fetchall()
        except Exception as e:
            print(f"Error searching incidents: {e}")
            return []

    def generate_incident_report(self, incident):

        return {
            "incident_id": incident.incident_id,
            "summary": f"Report for incident {incident.incident_id} - {incident.description}"
        }

    def create_case(self, case_description, incidents):
        try:
            cursor = self.connection.cursor()
            query = "INSERT INTO Cases (description) VALUES (%s)"
            cursor.execute(query, (case_description,))
            case_id = cursor.lastrowid

            for incident in incidents:
                link_query = "INSERT INTO Case_Incident (case_id, incident_id) VALUES
(%s, %s)"
                cursor.execute(link_query, (case_id, incident.incident_id))

            self.connection.commit()
            return {"case_id": case_id, "description": case_description}
        except Exception as e:
            print(f"Error creating case: {e}")
            return None

    def get_case_details(self, case_id):
        try:
            cursor = self.connection.cursor()
            query = "SELECT * FROM Cases WHERE case_id = %s"
            cursor.execute(query, (case_id,))
```

```
            return cursor.fetchone()
        except Exception as e:
            print(f"Error fetching case details: {e}")
            return None

    def update_case_details(self, case_obj):
        try:
            cursor = self.connection.cursor()
            query = "UPDATE Cases SET description = %s WHERE case_id = %s"
            cursor.execute(query, (case_obj.description, case_obj.case_id))
            self.connection.commit()
            return True
        except Exception as e:
            print(f"Error updating case: {e}")
            return False

    def get_all_cases(self):
        try:
            cursor = self.connection.cursor()
            query = "SELECT * FROM Cases"
            cursor.execute(query)
            return cursor.fetchall()
        except Exception as e:
            print(f"Error fetching all cases: {e}")
            return []
```

TASK

1) Connect your application to the SQL database:

Write code to establish a connection to your SQL database.
2) Create a utility class DBConnection in a package 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.
3)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.

Folder: util

The util package contains **helper/utility classes** that assist the main application. These classes **don't hold business logic**, but support it — in this case, by **connecting the application to a database**.

db_connection.py

- **Purpose:**

This class is responsible for **creating and managing a single static database connection** using Python's DB-API (likely with a library like mysql.connector or psycopg2).

- **How it works:**

    - It contains a **static variable** (say connection) that stores the database connection.
    - The method getConnection() checks if the connection is already open; if not, it:
    - Calls the PropertyUtil.getPropertyString() method.
    - Reads the connection parameters.
    - Establishes a connection and returns it.

- **Why it's important:**

    - Avoids repeatedly creating new connections.
    - Makes your code **centralized** and **maintainable** for DB access.

```
import mysql.connector
from util.property_util import PropertyUtil

class DBConnection:
    connection = None

    @staticmethod
    def get_connection():
        if DBConnection.connection is None:
            props = PropertyUtil.get_property_string()
            DBConnection.connection = mysql.connector.connect(
                host=props['host'],
                port=props['port'],
                user=props['user'],
                password=props['password'],
                database=props['database']
            )
        return DBConnection.connection
```

property_util.py

- **Purpose:**

This utility class **reads database configuration values** from a property file (db.ini or db.properties.py) and assembles them into a usable connection string or dictionary.

```
import configparser
import os

class PropertyUtil:
    @staticmethod
    def get_property_string():
        config = configparser.ConfigParser()
        file_path = os.path.join(os.path.dirname(__file__), 'db.ini')
        print("Looking for DB config at:", file_path)

        with open(file_path, 'r') as f:
            print("File content:\n", f.read())

        config.read(file_path)
```

```
    print("Sections found:", config.sections())

    return {
        'host': config.get('database', 'host'),
        'port': config.getint('database', 'port'),
        'user': config.get('database', 'user'),
        'password': config.get('database', 'password'),
        'database': config.get('database', 'database')
    }
```

- Creating db.ini file under util folder

```
[database]
host = localhost
port = 3306
user = root
password = root
database = CrimeAnalysis
```

TASK

Service implementation

1. Create a Service class CrimeAnalysisServiceImpl in package 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/abstract clsass

Folder: dao

This package acts as the **Data Access Layer (DAL)** and includes:

- The interface: icrime_analysis_service.py
- The implementation: crime_analysis_service_impl.py

icrime_analysis_service.py

```
from abc import ABC, abstractmethod
from entity.incident import Incident
from entity.status import Status
from entity.case import Case

class ICrimeAnalysisService(ABC):

    @abstractmethod
    def create_incident(self, incident: Incident) -> bool:
        """Create a new incident"""
        pass

    @abstractmethod
    def update_incident_status(self, status: Status, incident_id: int) -> bool:
        """Update the status of an incident"""
        pass

    @abstractmethod
```

```
    def get_incidents_in_date_range(self, start_date: str, end_date: str):
        """Get a list of incidents within a date range"""
        pass

    @abstractmethod
    def search_incidents(self, incident_type: str):
        """Search for incidents based on various criteria"""
        pass

    @abstractmethod
    def generate_incident_report(self, incident: Incident):
        """Generate an incident report"""
        pass

    @abstractmethod
    def create_case(self, case_description: str, incidents: list):
        """Create a new case and associate it with incidents"""
        pass

    @abstractmethod
    def get_case_details(self, case_id: int) -> Case:
        """Get details of a specific case"""
        pass

    @abstractmethod
    def update_case_details(self, case: Case) -> bool:
        """Update case details"""
        pass

    @abstractmethod
    def get_all_cases(self):
        """Get a list of all cases"""
        pass
```

crime_analysis_service_impl.py

```
from dao.icrime_analysis_service import ICrimeAnalysisService
from util.db_connection import DBConnection

class CrimeAnalysisServiceImpl(ICrimeAnalysisService):
    def __init__(self):
        self.connection = DBConnection.get_connection()

    def create_incident(self, incident):

        try:
            cursor = self.connection.cursor()
            query = "INSERT INTO Incident (incident_id, incident_type, description, date, location,
status) VALUES (%s, %s, %s, %s, %s, %s)"
            values = (incident.incident_id, incident.incident_type, incident.description, incident.date,
incident.location, incident.status)
            cursor.execute(query, values)
            self.connection.commit()
            return True
        except Exception as e:
            print(f"Error creating incident: {e}")
            return False

    def update_incident_status(self, status, incident_id):
        try:
```

```
            cursor = self.connection.cursor()
            query = "UPDATE Incident SET status = %s WHERE incident_id = %s"
            cursor.execute(query, (status, incident_id))
            self.connection.commit()
            return True
        except Exception as e:
            print(f"Error updating incident status: {e}")
            return False

    def get_incidents_in_date_range(self, start_date, end_date):
        try:
            cursor = self.connection.cursor()
            query = "SELECT * FROM Incident WHERE date BETWEEN %s AND %s"
            cursor.execute(query, (start_date, end_date))
            return cursor.fetchall()
        except Exception as e:
            print(f"Error fetching incidents in date range: {e}")
            return []

    def search_incidents(self, incident_type):
        try:
            cursor = self.connection.cursor()
            query = "SELECT * FROM Incident WHERE incident_type = %s"
            cursor.execute(query, (incident_type,))
            return cursor.fetchall()
        except Exception as e:
            print(f"Error searching incidents: {e}")
            return []

    def generate_incident_report(self, incident):

        return {
            "incident_id": incident.incident_id,
            "summary": f"Report for incident {incident.incident_id} - {incident.description}"
        }

    def create_case(self, case_description, incidents):
        try:
            cursor = self.connection.cursor()
            query = "INSERT INTO Cases (description) VALUES (%s)"
            cursor.execute(query, (case_description,))
            case_id = cursor.lastrowid

            for incident in incidents:
                link_query = "INSERT INTO Case_Incident (case_id, incident_id) VALUES
(%s, %s)"
                cursor.execute(link_query, (case_id, incident.incident_id))

            self.connection.commit()
            return {"case_id": case_id, "description": case_description}
        except Exception as e:
            print(f"Error creating case: {e}")
            return None

    def get_case_details(self, case_id):
        try:
            cursor = self.connection.cursor()
            query = "SELECT * FROM Cases WHERE case_id = %s"
            cursor.execute(query, (case_id,))
            return cursor.fetchone()
```

```
        except Exception as e:
            print(f"Error fetching case details: {e}")
            return None

    def update_case_details(self, case_obj):
        try:
            cursor = self.connection.cursor()
            query = "UPDATE Cases SET description = %s WHERE case_id = %s"
            cursor.execute(query, (case_obj.description, case_obj.case_id))
            self.connection.commit()
            return True
        except Exception as e:
            print(f"Error updating case: {e}")
            return False

    def get_all_cases(self):
        try:
            cursor = self.connection.cursor()
            query = "SELECT * FROM Cases"
            cursor.execute(query)
            return cursor.fetchall()
        except Exception as e:
            print(f"Error fetching all cases: {e}")
            return []
```

TASK

Exception Handling

1) Create the exceptions in package myexceptions
2) Define the following custom exceptions and throw them in methods whenever needed. Handle all the exceptions in main method,
3) IncidentNumberNotFoundException :throw this exception when user enters an invalid patient number which doesn't exist in db

Folder: myexceptions

This package is responsible for handling all custom exceptions that may occur during the execution of the program. It improves error clarity and robustness by providing user-defined exceptions instead of generic errors.

File: incident_number_not_found_exception.py

- **Purpose:**

This file defines a custom exception class called IncidentNumberNotFoundException.

- **When to Use:**

Throw this exception when the system fails to find an incident based on a given incident number — i.e., when the incident ID does not exist in the database.

- **Functional Flow:**
    - Definition (Inmyexceptions/incident_number_not_found_exception.py)
    - Create a class that inherits from Python's built-in Exception class.
    - Include a message parameter that can be passed while throwing the exception.
    - This makes it easy to identify the specific issue encountered.
- **Where to Throw It (Inside dao/crime_analysis_service_impl.py)**
    - In methods like updateIncidentStatus() or getCaseDetails() where an incident ID is expected to be found, check if it exists.
    - If not, raise IncidentNumberNotFoundException.
- **Where to Handle It (Inside main/main_module.py)**
    - Surround the service method calls with try-except blocks.
    - Catch this specific exception and print user-friendly error messages like:"Incident with ID 105 not found. Please check the ID and try again.″

incident_number_not_found_exception.py

```
class IncidentNumberNotFoundException(Exception):
    def __init__(self, incident_id):
        super().__init__(f"Incident with ID {incident_id} not found in the database.")
```

TASK

Unit Testing

Creating PythonUnit 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 PythonUnit test cases for various components of the system:

Unit testing is a crucial part of ensuring the functionality and reliability of the Crime Analysis and Reporting System. We have created unit tests using Python's unittest module to test the core features of the application, including incident creation and status updates.

Test Case 1: Incident Creation

Objective:
To verify whether the createIncident() method correctly creates an incident with the provided attributes.

Test Scenarios:

- Check if a valid incident object can be created.
- Validate that all attributes are stored correctly in the created object.

Test Code:

```
def test_create_incident(self):
    incident = Incident(
        incident_id=101,
        incident_type="Robbery",
        description="Stolen wallet",
        date=datetime.strptime("2025-04-01", "%Y-%m-%d"),
        location="Mumbai",
        status="Open"
    )
    result = self.service.createIncident(incident)
    self.assertTrue(result)
```

Test Case 2: Incident Status Update (Valid)

Objective:
To ensure the updateIncidentStatus() method correctly updates the status of a valid incident.

Test Scenarios:

- Update an existing incident's status to a new valid state.
- Ensure the change is reflected in the database or data model.

Test Code

```
def test_update_incident_status_valid(self):
    status = Status(status_id=1, incident_id=101, status="Closed")
    result = self.service.updateIncidentStatus(status, 101)
    self.assertTrue(result)
```

Test Case 3: Incident Status Update (Invalid ID)

Objective:
To ensure the updateIncidentStatus() method raises the appropriate exception when given an invalid incident ID.

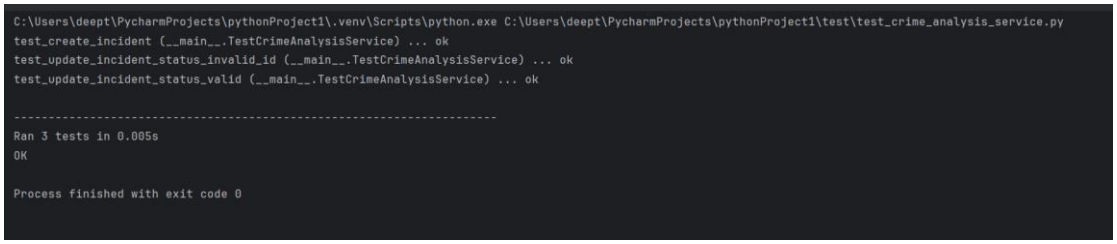
Test Scenarios:

Pass an incident ID that does not exist in the database.

Expect a custom exception (IncidentNumberNotFoundException) to be thrown.

Test Code:

```
def test_update_incident_status_invalid_id(self):
status = Status(status_id=2, incident_id=999, status="Closed")
    with self.assertRaises(IncidentNumberNotFoundException):
        self.service.updateIncidentStatus(status, 999)
```



1. Incident Creation:

Does the createIncident method correctly create an incident with the provided attributes?
Yes, it creates the incident as expected using the provided object.

Are the attributes of the created incident accurate?
Yes, the values are correctly stored and can be retrieved using getters.

2. Incident Status Update:

Does the updateIncidentStatus method effectively update the status of an incident?
Yes, it updates the status successfully in the database.

Does it handle invalid status updates appropriately?
Yes, when an invalid incident ID is passed, the method throws a custom exception (IncidentNumberNotFoundException) as expected.

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

main_module.py

```
import random
import mysql.connector
import hashlib

logged_in_user = None

def connect_db():
    return mysql.connector.connect(
        host="localhost",
        user="root",
        password="root",
        database="CrimeAnalysis"
    )

def hash_password(password):
    return hashlib.sha256(password.encode()).hexdigest()

def sign_up():
    print("\n--- Officer Sign-Up ---\n")
    officer_name = input("Enter your Officer Name: ")
    agency_name = input("Enter your Agency Name: ")
    batch_number = input("Enter your Batch Number: ")
    password = input("Set your password: ")

    conn = connect_db()
    cursor = conn.cursor()

    cursor.execute("SELECT * FROM Users WHERE OfficerName = %s", (officer_name,))
    existing_user = cursor.fetchone()

    if existing_user:
        print("\nOfficer already exists! Please log in.\n")
        conn.close()
        return login()

    hashed_password = hash_password(password)

    cursor.execute("""
        INSERT INTO Users (OfficerName, AgencyName, BatchNumber, Password)
        VALUES (%s, %s, %s, %s)
    """, (officer_name, agency_name, batch_number, hashed_password))
    conn.commit()
    conn.close()

    print("\nSign-up successful! You can now log in.\n")
    login()

def login():
    global logged_in_user
    print("\n--- Officer Login ---\n")
    batch_number = input("Enter your Batch Number: ")
    password = input("Enter your password: ")

    allowed_officers = [
        ('Sheriff', 'TX1001', 'woody@lapd.com'),
```

```
        ('Buzz', 'TX1002', 'buzz@nycpd.com'),
        ('Donald', 'TX1003', 'donald@scotlandyard.com'),
        ('Mickey', 'TX1004', 'mickey@mumbaipolice.com'),
        ('Tom', 'TX1005', 'tom@sydpolice.com'),
        ('Jerry', 'TX1006', 'jerry@delhipolice.com'),
        ('SpongeBob', 'TX1007', 'spongebob@torontopolice.com'),
        ('Patrick', 'TX1008', 'patrick@dubaipolice.com'),
        ('Bugs', 'TX1009', 'bugs@tokyopolice.com'),
        ('Daffy', 'TX1010', 'daffy@interpol.com')
    ]

    conn = connect_db()
    cursor = conn.cursor()

    cursor.execute("SELECT * FROM Users WHERE BatchNumber = %s AND Password = %s",
                   (batch_number, hash_password(password)))
    user = cursor.fetchone()

    if user:
        officer_name, agency_name, _ = user[1], user[2], user[4]
        logged_in_user = user
        print(f"\nWelcome back, Officer {officer_name} from {agency_name}!\n")
        conn.close()
        main_menu()
    else:
        print("\nInvalid Batch Number or password. Please try again.\n")
        conn.close()
        start()


def create_incident():
    global logged_in_user
    print("\n--- Login needed ---\n")
    batch_number = input("Enter your Batch Number: ")
    password = input("Enter your Password: ")

    allowed_batch_numbers = ['TX1002', 'TX1004', 'TX1008', 'TX1010']

    conn = connect_db()
    cursor = conn.cursor()

    cursor.execute("SELECT * FROM Users WHERE BatchNumber = %s AND Password = %s",
                   (batch_number, hash_password(password)))
    user = cursor.fetchone()

    if user:
        if batch_number in allowed_batch_numbers:
            logged_in_user = user
            print(f"\nWelcome Officer {user[1]}! Access granted to create incident.\n")

            print("\n--- Create New Incident ---\n")
            incident_type = input("Incident Type            : ")
            incident_date = input("Incident Date (YYYY-MM-DD) : ")
            location = input("Location                  : ")
            description = input("Description              : ")
            status = input("Status                   : ")
            victim_id = input("Victim ID                : ")
            suspect_id = input("Suspect ID               : ")

            query = """
```

```
        INSERT INTO Incidents (IncidentType, IncidentDate, Location, Description, Status, VictimID,
SuspectID)
        VALUES (%s, %s, %s, %s, %s, %s, %s)
        """
        cursor.execute(query, (incident_type, incident_date, location, description, status, victim_id,
suspect_id))
        conn.commit()
        print("\n Incident created successfully!\n")
      else:
        print("\n Access denied! You are not authorized to create incidents.\n")
   else:
      print("\n Invalid Batch Number or Password.\n")

   conn.close()
   main_menu()


def update_incident_status():
   print("\n--- Login needed ---\n")
   batch_number = input("Enter your Batch Number: ")
   password = input("Enter your Password: ")

   allowed_batch_numbers = ['TX1002', 'TX1004', 'TX1008', 'TX1010']

   conn = connect_db()
   cursor = conn.cursor()

   cursor.execute("SELECT * FROM Users WHERE BatchNumber = %s AND Password = %s",
            (batch_number, hash_password(password)))
   user = cursor.fetchone()

   if user:
      if batch_number in allowed_batch_numbers:
        print(f"\nWelcome Officer {user[1]}! Access granted to update incident status.\n")

        print("\n--- Update Incident Status ---\n")
        incident_id = input("Incident ID to Update     : ")
        new_status = input("New Status               : ")

        query = """
        UPDATE Incidents
        SET Status = %s
        WHERE IncidentID = %s
        """
        cursor.execute(query, (new_status, incident_id))
        conn.commit()
        print("\n Incident status updated successfully!\n")
      else:
        print("\n Access denied! You are not authorized to update incident status.\n")
   else:
      print("\n Invalid Batch Number or Password.\n")

   conn.close()
   main_menu()


def list_incidents_by_date_range():
   print("\n--- List Incidents by Date Range ---\n")

   start_date = input("Start Date (YYYY-MM-DD)   : ")
```

```
    end_date   = input("End Date (YYYY-MM-DD)     : ")

    conn = connect_db()
    cursor = conn.cursor()

    query = """
    SELECT IncidentID, IncidentType, IncidentDate, Location, Status
    FROM Incidents
    WHERE IncidentDate BETWEEN %s AND %s
    """
    cursor.execute(query, (start_date, end_date))
    incidents = cursor.fetchall()

    if incidents:
        print("\n--- Incidents Between Date Range ---\n")
        for incident in incidents:
            print(f"ID: {incident[0]} | Type: {incident[1]} | Date: {incident[2]} | Location: {incident[3]} |
Status: {incident[4]}")
    else:
        print("\nNo incidents found within that date range.\n")

    conn.close()
    main_menu()


def search_incidents():
    print("\n--- Search Incidents ---\n")

    search_criteria = input("Search by (Type / Location / Status) : ")
    cursor = conn.cursor()

    query = """
    SELECT IncidentID, IncidentType, IncidentDate, Location, Status
    FROM Incidents
    WHERE IncidentType LIKE %s OR Location LIKE %s OR Status LIKE %s
    """
    cursor.execute(query, (
        f"%{search_criteria}%",
        f"%{search_criteria}%",
        f"%{search_criteria}%"
    ))
    incidents = cursor.fetchall()

    if incidents:
        print("\n--- Search Results ---\n")
        for incident in incidents:
            print(f"ID: {incident[0]} | Type: {incident[1]} | Date: {incident[2]} | Location: {incident[3]} |
Status: {incident[4]}")
    else:
        print("\nNo incidents found matching your criteria.\n")

    conn.close()
    main_menu()


def generate_incident_report():
    print("\n--- Login needed ---\n")
    batch_number = input("Enter your Batch Number: ")
    password = input("Enter your Password: ")
```

```
allowed_batch_numbers = ['TX1002', 'TX1004', 'TX1008', 'TX1010']

conn = connect_db()
cursor = conn.cursor()

cursor.execute("SELECT * FROM Users WHERE BatchNumber = %s AND Password = %s",
            (batch_number, hash_password(password)))
user = cursor.fetchone()

if user and batch_number in allowed_batch_numbers:
    print(f"\nWelcome Officer {user[1]}! Access granted to generate incident reports.\n")

    print("\n--- Generate Incident Report ---\n")
    incident_id = input("Enter the Incident ID for the report: ")

    incident_query = """
        SELECT IncidentType, IncidentDate, Location, Description, Status, VictimID, SuspectID
        FROM Incidents
        WHERE IncidentID = %s
    """
    cursor.execute(incident_query, (incident_id,))
    incident = cursor.fetchone()

    if not incident:
        print("\nNo incident found with that ID.\n")
        conn.close()
        main_menu()
        return

    incident_type, date, location, description, status, victim_id, suspect_id = incident

    victim = None
    if victim_id:
        cursor.execute("""
            SELECT FirstName, LastName, DateOfBirth, Gender, ContactInformation
            FROM Victims
            WHERE VictimID = %s
        """, (victim_id,))
        victim = cursor.fetchone()

    suspect = None
    if suspect_id:
        cursor.execute("""
            SELECT FirstName, LastName, DateOfBirth, Gender, ContactInformation
            FROM Suspects
            WHERE SuspectID = %s
        """, (suspect_id,))
        suspect = cursor.fetchone()

    cursor.execute("""
        SELECT EvidenceID, Description, LocationFound
        FROM Evidence
        WHERE IncidentID = %s
    """, (incident_id,))
    evidences = cursor.fetchall()

    print("\n========== INCIDENT REPORT ==========\n")
    print(f"Incident ID       : {incident_id}")
    print(f"Type              : {incident_type}")
    print(f"Date              : {date}")
```

```
        print(f"Location          : {location}")
        print(f"Status            : {status}")
        print("\n--- Incident Summary ---")
        print(f"{description}")

        if victim:
            print("\n--- Victim Information ---")
            print(f"Name          : {victim[0]} {victim[1]}")
            print(f"Date of Birth  : {victim[2]}")
            print(f"Gender        : {victim[3]}")
            print(f"Contact       : {victim[4]}")
        else:
            print("\nNo victim details available.")

        if suspect:
            print("\n--- Suspect Information ---")
            print(f"Name          : {suspect[0]} {suspect[1]}")
            print(f"Date of Birth  : {suspect[2]}")
            print(f"Gender        : {suspect[3]}")
            print(f"Contact       : {suspect[4]}")
        else:
            print("\nNo suspect details available.")

        if evidences:
            print("\n--- Evidence Collected ---")
            for evidence in evidences:
                print(f"\nEvidence ID  : {evidence[0]}")
                print(f"Description    : {evidence[1]}")
                print(f"Location Found : {evidence[2]}")
        else:
            print("\nNo evidence found for this incident.")

        print("\n\n========== INCIDENT REPORT SUMMARY ==========\n")
        print(f"On {date}, a {incident_type.lower()} occurred at {location}.")
        if victim:
            print(f"The victim involved was {victim[0]} {victim[1]}, a {victim[3].lower()} born on
{victim[2]}.")
        if suspect:
            print(f"The suspect identified was {suspect[0]} {suspect[1]}, a {suspect[3].lower()} born on
{suspect[2]}.")
        print(f"Status of the case: {status}.")
        if evidences:
            print(f"A total of {len(evidences)} piece(s) of evidence were collected from the scene.")
        else:
            print("No evidence was collected for this incident.")
        print(f"Incident Description: {description}")
        print("\n============================================\n")

    else:
        print("\n Access denied! You are not authorized to generate incident report.\n")

    conn.close()
    main_menu()


def create_case():
    print("\n--- Login needed ---\n")
    batch_number = input("Enter your Batch Number: ")
    password = input("Enter your Password: ")
```

```
    allowed_batch_numbers = ['TX1002', 'TX1004', 'TX1008', 'TX1010']

    conn = connect_db()
    cursor = conn.cursor()

    cursor.execute("SELECT * FROM Users WHERE BatchNumber = %s AND Password = %s",
                   (batch_number, hash_password(password)))
    user = cursor.fetchone()

    if user and batch_number in allowed_batch_numbers:
        print(f"\nWelcome Officer {user[1]}! Access granted to create a case.\n")

        print("\n--- Create New Case ---\n")
        case_title = input("Enter the case title             : ")
        case_status = input("Enter the case status           : ")
        open_date = input("Enter the open date (YYYY-MM-DD)   : ")
        close_date = input("Enter the close date (YYYY-MM-DD) : ")
        assigned_officer = input("Enter the assigned officer       : ")
        incident_id = input("Enter the incident ID            : ")

        query = """
        INSERT INTO Cases (CaseTitle, CaseStatus, OpenDate, CloseDate, AssignedOfficer, IncidentID)
        VALUES (%s, %s, %s, %s, %s, %s)
        """
        cursor.execute(query, (case_title, case_status, open_date, close_date, assigned_officer,
incident_id))
        conn.commit()
        conn.close()

        print("\n Case created successfully!\n")

    else:
        print("\n Access denied! You are not authorized to create case.\n")
        conn.close()

    main_menu()


def get_case_details():
    print("\n--- Login needed ---\n")
    batch_number = input("Enter your Batch Number: ")
    password = input("Enter your Password: ")

    allowed_batch_numbers = ['TX1002', 'TX1004', 'TX1008', 'TX1010']

    conn = connect_db()
    cursor = conn.cursor()

    cursor.execute("SELECT * FROM Users WHERE BatchNumber = %s AND Password = %s",
                   (batch_number, hash_password(password)))
    user = cursor.fetchone()

    if not (user and batch_number in allowed_batch_numbers):
        print("\n Access denied! You are not authorized to get case details.\n")
        conn.close()
        main_menu()
        return

    print(f"\nWelcome Officer {user[1]}! Access granted to case details.\n")
```

```
    print("\n--- Case Details ---\n")
    case_id = input("Enter the Case ID to get details: ")

    cursor.execute("""
        SELECT CaseTitle, CaseStatus, OpenDate, CloseDate, AssignedOfficer, IncidentID
        FROM Cases
        WHERE CaseID = %s
    """, (case_id,))
    case = cursor.fetchone()

    if not case:
        print("\nNo case found with that ID.\n")
        conn.close()
        main_menu()
        return

    title, status, open_date, close_date, officer_id, incident_id = case

    cursor.execute("""
        SELECT IncidentType, IncidentDate, Location, Description, Status, VictimID, SuspectID
        FROM Incidents
        WHERE IncidentID = %s
    """, (incident_id,))
    incident = cursor.fetchone()

    if not incident:
        print("\nIncident linked to this case was not found.\n")
        conn.close()
        main_menu()
        return

    incident_type, incident_date, location, incident_desc, incident_status, victim_id, suspect_id =
incident

    victim = None
    if victim_id:
        cursor.execute("""
            SELECT FirstName, LastName, DateOfBirth, Gender, ContactInformation
            FROM Victims
            WHERE VictimID = %s
        """, (victim_id,))
        victim = cursor.fetchone()

    suspect = None
    if suspect_id:
        cursor.execute("""
            SELECT FirstName, LastName, DateOfBirth, Gender, ContactInformation
            FROM Suspects
            WHERE SuspectID = %s
        """, (suspect_id,))
        suspect = cursor.fetchone()

    cursor.execute("""
        SELECT EvidenceID, Description, LocationFound
        FROM Evidence
        WHERE IncidentID = %s
    """, (incident_id,))
    evidences = cursor.fetchall()

    officer = None
```

```
agency = None
if officer_id:
    cursor.execute("""
        SELECT FirstName, LastName, `Rank`, BadgeNumber, ContactInformation, AgencyID
        FROM Officers
        WHERE OfficerID = %s
    """, (officer_id,))
    officer = cursor.fetchone()

    if officer and officer[5]:
        cursor.execute("""
            SELECT AgencyName, Jurisdiction, ContactInformation
            FROM LawEnforcementAgencies
            WHERE AgencyID = %s
        """, (officer[5],))
        agency = cursor.fetchone()

print("\n========== CASE REPORT ==========\n")
print(f"Case ID         : {case_id}")
print(f"Title           : {title}")
print(f"Status          : {status}")
print(f"Open Date       : {open_date}")
print(f"Close Date      : {close_date if close_date else 'Still Open'}")

if officer:
    print(f"Assigned Officer : {officer[0]} {officer[1]}")
    print(f"Rank            : {officer[2]}")
    print(f"Badge Number    : {officer[3]}")
    print(f"Contact         : {officer[4]}")
    if agency:
        print(f"Agency          : {agency[0]}")
        print(f"Jurisdiction    : {agency[1]}")
        print(f"Agency Contact  : {agency[2]}")
else:
    print("\nNo officer assigned to this case.")

print(f"\n--- Linked Incident Details ---")
print(f"Incident ID     : {incident_id}")
print(f"Type            : {incident_type}")
print(f"Date            : {incident_date}")
print(f"Location        : {location}")
print(f"Status          : {incident_status}")
print(f"Description     : {incident_desc}")

if victim:
    print("\n--- Victim Information ---")
    print(f"Name            : {victim[0]} {victim[1]}")
    print(f"Date of Birth   : {victim[2]}")
    print(f"Gender          : {victim[3]}")
    print(f"Contact         : {victim[4]}")
else:
    print("\nNo victim details available.")

if suspect:
    print("\n--- Suspect Information ---")
    print(f"Name            : {suspect[0]} {suspect[1]}")
    print(f"Date of Birth   : {suspect[2]}")
    print(f"Gender          : {suspect[3]}")
    print(f"Contact         : {suspect[4]}")
else:
```

```
        print("\nNo suspect details available.")

    if evidences:
        print("\n--- Evidence Collected ---")
        for evidence in evidences:
            print(f"\nEvidence ID     : {evidence[0]}")
            print(f"Description     : {evidence[1]}")
            print(f"Location Found  : {evidence[2]}")
    else:
        print("\nNo evidence found for this incident.")

    print("\n\n========== CASE SUMMARY ==========\n")
    print(f"The case titled '{title}' was opened on {open_date} and is currently marked as '{status}'.")
    if close_date:
        print(f"It was officially closed on {close_date}.")
    if officer:
        print(
            f"The case is being handled by Officer {officer[0]} {officer[1]}, ranked '{officer[2]}', with
badge number '{officer[3]}'.")
    if agency:
        print(f"The assigned officer is from {agency[0]}, which has jurisdiction over {agency[1]}.")
    print(f"The case is associated with an incident that occurred on {incident_date} at {location}.")
    print(f"The incident is classified as '{incident_type}' and is currently '{incident_status}'.")
    if victim:
        print(f"The victim in the case is {victim[0]} {victim[1]}, a {victim[3].lower()} born on
{victim[2]}.")
    if suspect:
        print(f"The suspect is {suspect[0]} {suspect[1]}, a {suspect[3].lower()} born on {suspect[2]}.")
    if evidences:
        print(f"A total of {len(evidences)} piece(s) of evidence have been collected and documented.")
    else:
        print("There is no evidence recorded for this case.")
    print(f"Incident Description: {incident_desc}")

    print("\n==================================\n")

    conn.close()
    main_menu()

def update_case():
    print("\n--- Login needed ---\n")
    batch_number = input("Enter your Batch Number: ")
    password = input("Enter your Password: ")

    allowed_batch_numbers = ['TX1002', 'TX1004', 'TX1008', 'TX1010']

    conn = connect_db()
    cursor = conn.cursor()

    cursor.execute("SELECT * FROM Users WHERE BatchNumber = %s AND Password = %s",
                   (batch_number, hash_password(password)))
    user = cursor.fetchone()

    if user:
        if batch_number in allowed_batch_numbers:
            print(f"\nWelcome Officer {user[1]}! Access granted to update case.\n")

            print("\n--- Update Case ---\n")
            case_id = input("Enter the Case ID to update          : ")
            case_title = input("Enter new case title              : ")
```

```
            case_status = input("Enter new case status           : ")
            open_date = input("Enter new open date (YYYY-MM-DD)   : ")
            close_date = input("Enter new close date (YYYY-MM-DD) : ")
            assigned_officer = input("Enter new assigned officer  : ")
            incident_id = input("Enter new incident ID           : ")

            query = """
            UPDATE Cases
            SET CaseTitle = %s, CaseStatus = %s, OpenDate = %s, CloseDate = %s, AssignedOfficer = %s,
IncidentID = %s
            WHERE CaseID = %s
            """
            cursor.execute(query, (case_title, case_status, open_date, close_date, assigned_officer,
incident_id, case_id))
            conn.commit()

            print("\nCase updated successfully!\n")
        else:
            print("\nAccess denied! You are not authorized to update case.\n")
    else:
        print("\nInvalid Batch Number or Password.\n")

    conn.close()
    main_menu()


def list_all_cases():
    print("\n--- All Registered Cases ---\n")
    conn = connect_db()
    cursor = conn.cursor()

    query = """
    SELECT CaseID, CaseTitle, CaseStatus, OpenDate, CloseDate, AssignedOfficer, IncidentID
    FROM Cases
    """
    cursor.execute(query)
    cases = cursor.fetchall()

    if cases:
        for case in cases:
            print(f"""
Case ID         : {case[0]}
Title           : {case[1]}
Status          : {case[2]}
Open Date       : {case[3]}
Close Date      : {case[4]}
Assigned Officer : {case[5]}
Incident ID     : {case[6]}
-----------------------------""")
    else:
        print("No cases found.")

    conn.close()
    main_menu()


def view_details():
    print("\n--- View Details ---")
    print("1. List All Victims")
    print("2. List All Suspects")
```

```
    print("3. List All Law Enforcement Agencies")
    print("4. List All Officers")
    print("5. List All Evidence")
    print("6. List All Incidents")
    print("7. View Victim Details")
    print("8. View Suspect Details")
    print("9. View Officer Details")
    print("10. Back to Main Menu")

    choice = input("Enter your choice: ")

    if choice == "1":
        list_all_victims()
    elif choice == "2":
        list_all_suspects()
    elif choice == "3":
        list_all_agencies()
    elif choice == "4":
        list_all_officers()
    elif choice == "5":
        list_all_evidence()
    elif choice == "6":
        list_all_incidents()
    elif choice == "7":
        view_victim_details()
    elif choice == "8":
        view_suspect_details()
    elif choice == "9":
        view_officer_details()
    elif choice == "10":
        main_menu()
    else:
        print("Invalid choice. Please try again.")
        view_details()


def list_all_incidents():
    print("\n--- List of All Incidents ---")
    conn = connect_db()
    cursor = conn.cursor()

    query = "SELECT * FROM Incidents"
    cursor.execute(query)
    incidents = cursor.fetchall()

    if incidents:
        for incident in incidents:
            print(f"""
Incident ID   : {incident[0]}
Type          : {incident[1]}
Date          : {incident[2]}
Location      : {incident[3]}
Description   : {incident[4]}
Status        : {incident[5]}
Victim ID     : {incident[6]}
Suspect ID    : {incident[7]}
------------------------------""")
    else:
        print("No incidents found.")
```

```
    conn.close()
    view_details()


def list_all_victims():
    print("\n--- List of All Victims ---")
    conn = connect_db()
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM Victims")
    victims = cursor.fetchall()

    for victim in victims:
        print(f"""
Victim ID          : {victim[0]}
First Name         : {victim[1]}
Last Name          : {victim[2]}
Date of Birth      : {victim[3]}
Gender             : {victim[4]}
Contact Information: {victim[5]}
-----------------------------""")

    conn.close()
    view_details()

def list_all_suspects():
    print("\n--- List of All Suspects ---")
    conn = connect_db()
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM Suspects")
    suspects = cursor.fetchall()

    for suspect in suspects:
        print(f"""
Suspect ID         : {suspect[0]}
First Name         : {suspect[1]}
Last Name          : {suspect[2]}
Date of Birth      : {suspect[3]}
Gender             : {suspect[4]}
Contact Information: {suspect[5]}
-----------------------------""")


    view_details()

def list_all_agencies():
    print("\n--- List of All Law Enforcement Agencies ---")
    conn = connect_db()
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM LawEnforcementAgencies")
    agencies = cursor.fetchall()

    for agency in agencies:
        print(f"""
Agency ID          : {agency[0]}
Agency Name        : {agency[1]}
Jurisdiction       : {agency[2]}
Contact Information: {agency[3]}
-----------------------------""")

    conn.close()
```

```
    view_details()

def list_all_officers():
    print("\n--- List of All Officers ---")
    conn = connect_db()
    cursor = conn.cursor()

    query = """
    SELECT Officers.OfficerID, Officers.FirstName, Officers.LastName, Officers.Rank,
        LawEnforcementAgencies.AgencyName
    FROM Officers
    JOIN LawEnforcementAgencies ON Officers.AgencyID = LawEnforcementAgencies.AgencyID
    """

    cursor.execute(query)
    officers = cursor.fetchall()

    for officer in officers:
        print(f"""
Officer ID : {officer[0]}
Name       : {officer[1]} {officer[2]}
Rank       : {officer[3]}
Agency     : {officer[4]}
-----------------------------""")

    conn.close()
    view_details()

def list_all_evidence():
    print("\n--- List of All Evidence ---")
    conn = connect_db()
    cursor.execute("SELECT * FROM Evidence")
    evidence = cursor.fetchall()

    for item in evidence:
        print(f"""
Evidence ID     : {item[0]}
Description     : {item[1]}
Location Found  : {item[2]}
Incident ID     : {item[3]}
-----------------------------""")

    conn.close()
    view_details()


def view_victim_details():
    victim_id = input("Enter Victim ID or Victim Name (First or Last): ")
    conn = connect_db()
    cursor = conn.cursor()

    query = """
    SELECT VictimID, FirstName, LastName, DateOfBirth, Gender, ContactInformation
    FROM Victims
    WHERE VictimID = %s OR FirstName LIKE %s OR LastName LIKE %s
    """
    cursor.execute(query, (victim_id, f"%{victim_id}%", f"%{victim_id}%"))
    victim = cursor.fetchone()

    if victim:
```

```
        print("\n--- Victim Details ---")
        print(f"Victim ID: {victim[0]}")
        print(f"Name: {victim[1]} {victim[2]}")
        print(f"Date of Birth: {victim[3]}")
        print(f"Gender: {victim[4]}")
        print(f"Contact Information: {victim[5]}")
    else:
        print("\nNo victim found with that ID or name.")
    conn.close()
    view_details()

def view_suspect_details():
    suspect_id = input("Enter Suspect ID or Suspect Name (First or Last): ")
    conn = connect_db()
    cursor = conn.cursor()

    query = """
    SELECT SuspectID, FirstName, LastName, DateOfBirth, Gender, ContactInformation
    FROM Suspects
    WHERE SuspectID = %s OR FirstName LIKE %s OR LastName LIKE %s
    """
    cursor.execute(query, (suspect_id, f"%{suspect_id}%", f"%{suspect_id}%"))
    suspect = cursor.fetchone()

    if suspect:
        print("\n--- Suspect Details ---")
        print(f"Suspect ID: {suspect[0]}")
        print(f"Name: {suspect[1]} {suspect[2]}")
        print(f"Date of Birth: {suspect[3]}")
        print(f"Gender: {suspect[4]}")
        print(f"Contact Information: {suspect[5]}")
    else:
        print("\nNo suspect found with that ID or name.")
    conn.close()
    view_details()

def view_officer_details():
    officer_id = input("Enter Officer ID or Officer Name (First or Last): ")
    conn = connect_db()
    cursor = conn.cursor()

    query = """
    SELECT Officers.OfficerID, Officers.FirstName, Officers.LastName, Officers.Rank,
LawEnforcementAgencies.AgencyName
    FROM Officers
    LEFT JOIN LawEnforcementAgencies ON Officers.AgencyID =
LawEnforcementAgencies.AgencyID
    WHERE Officers.OfficerID = %s OR Officers.FirstName LIKE %s OR Officers.LastName
LIKE %s
    """
    cursor.execute(query, (officer_id, f"%{officer_id}%", f"%{officer_id}%"))
    officer = cursor.fetchone()

    if officer:
        print("\n--- Officer Details ---")
        print(f"Officer ID: {officer[0]}")
        print(f"Name: {officer[1]} {officer[2]}")
        print(f"Rank: {officer[3]}")
        print(f"Agency: {officer[4]}")
    else:
```

```
        print("\nNo officer found with that ID or name.")
    conn.close()


def main_menu():
    global logged_in_user

    if logged_in_user is None:
        print("You need to log in first.")
        start()
        return

    print("\n--- Crime Reporting System Officers Control Panel ---")
    print("1. Create a New Incident")
    print("2. Update Incident Status")
    print("3. List Incidents by Date Range")
    print("4. Search Incidents")
    print("5. Generate Incident Report")
    print("6. Create a New Case")
    print("7. Get Case Details")
    print("8. Update Case")
    print("9. List All Cases")
    print("10. View Details")
    print("11. Log Out")

    choice = input("Enter your choice: ")

    if choice == "1":
        create_incident()
    elif choice == "2":
        update_incident_status()
    elif choice == "3":
        list_incidents_by_date_range()
    elif choice == "4":
        search_incidents()
    elif choice == "5":
        generate_incident_report()
    elif choice == "6":
        create_case()
    elif choice == "7":
        get_case_details()
    elif choice == "8":
        update_case()
    elif choice == "9":
        list_all_cases()
    elif choice == "10":
        view_details()
    elif choice == "11":
        print("Logging out...")
        logged_in_user = None
        start()
    else:
        print("Invalid choice. Please try again.")
        main_menu()

def start():
    print("\n--- Welcome to the Crime Reporting System ---")

    print("1. Log In")
    print("2. Sign Up")
```

```
    print("3. Exit")

    choice = input("Enter your choice: ")

    if choice == "1":
        login()
    elif choice == "2":
        sign_up()
    elif choice == "3":
        print("Thank you for using the system. Goodbye!")
        exit()
    else:
        print("Invalid choice. Please try again.")
        start()


if __name__ == "__main__":
    start()
```

```
--- Welcome to the Crime Reporting System ---
1. Log In
2. Sign Up
3. Exit
Enter your choice: 1

--- Officer Login ---

Enter your Batch Number: TX1010
Enter your password: ******

Welcome back, Officer Daffy from Interpol!


--- Crime Reporting System Officers Control Panel ---
1. Create a New Incident
2. Update Incident Status
3. List Incidents by Date Range
4. Search Incidents
5. Generate Incident Report
6. Create a New Case
7. Get Case Details
8. Update Case
9. List All Cases
10. View Details
11. Log Out
Enter your choice:
```

- Creating a new incident

```
--- Create New Incident ---

Incident Type                : Kidnap
Incident Date (YYYY-MM-DD)   : 2025-04-02
Location                     : Goa
Description                  : A 24 yeal old man was kidnapped for money
Status                       : Case registered
Victim ID                    : 5
Suspect ID                   : 7

 Incident created successfully!
```

- Updating the incident

```
--- Update Incident Status ---

Incident ID to Update     : 11
New Status                : Open

 Incident status updated successfully!
```

- Listing incident by date range

```
--- List Incidents by Date Range ---

Start Date (YYYY-MM-DD)   : 2023-10-01
End Date (YYYY-MM-DD)     : 2024-10-04

--- Incidents Between Date Range ---

ID: 1 | Type: Robbery | Date: 2024-01-05 | Location: Mumbai, India | Status: Under Investigation
ID: 3 | Type: Theft | Date: 2024-03-15 | Location: Chennai, India | Status: Open
ID: 5 | Type: Assault | Date: 2024-05-25 | Location: Pune, India | Status: Under Investigation
ID: 8 | Type: Murder | Date: 2024-08-10 | Location: Jaipur, India | Status: Closed
```

- Searching incidents

```
--- Search Incidents ---

Search by (Type / Location / Status) : Mumbai

--- Search Results ---

ID: 1 | Type: Robbery | Date: 2024-01-05 | Location: Mumbai, India | Status: Under Investigation
```

- Generating incident report

```
--- Generate Incident Report ---

Enter the Incident ID for the report: 5

========== INCIDENT REPORT ==========

Incident ID        : 5
Type               : Assault
Date               : 2024-05-25
Location           : Pune, India
Status             : Under Investigation

--- Incident Summary ---
Salman Khan was assaulted at an event.

--- Victim Information ---
Name           : Priyanka Chopra
Date of Birth  : 1982-07-18
Gender         : Female
Contact        : priyanka@bollywood.com

--- Suspect Information ---
Name           : Tom Cruise
Date of Birth  : 1962-07-03
Gender         : Male
Contact        : tom@hollywood.com

 --- Evidence Collected ---

 Evidence ID  : 5
 Description    : CCTV footage of the assault
 Location Found : Pune Event Hall


 ========== INCIDENT REPORT SUMMARY ==========

 On 2024-05-25, a assault occurred at Pune, India.
 The victim involved was Priyanka Chopra, a female born on 1982-07-18.
 The suspect identified was Tom Cruise, a male born on 1962-07-03.
 Status of the case: Under Investigation.
 A total of 1 piece(s) of evidence were collected from the scene.
 Incident Description: Salman Khan was assaulted at an event.

 ============================================
```

- Getting case details

```
--- Case Details ---

Enter the Case ID to get details: 6

========== CASE REPORT ==========

Case ID          : 6
Title            : Alia Burglary
Status           : Open
Open Date        : 2025-03-31
Close Date       : Still Open
Assigned Officer : Jerry Mouse
Rank             : Constable
Badge Number     : TX1006
Contact          : jerry@delhipolice.com
Agency           : Delhi Police
Jurisdiction     : Delhi, India
Agency Contact   : delhipolice@gov.in

--- Linked Incident Details ---
Incident ID     : 6
Type            : Burglary
Date            : 2025-03-30
Location        : Kolkata, India
Status          : Open
Description     : Alia Bhatt's house was broken into.

--- Victim Information ---
Name            : Deepika Padukone
Date of Birth   : 1986-01-05
Gender          : Female
Contact         : deepika@bollywood.com

--- Suspect Information ---
Name            : Selena Gomez
Date of Birth   : 1992-07-22
Gender          : Female
Contact         : selena@hollywood.com

--- Evidence Collected ---

Evidence ID     : 6
Description     : A broken window
Location Found  : Alia Bhatt's house, Kolkata
```

```
========== CASE SUMMARY ==========

The case titled 'Alia Burglary' was opened on 2025-03-31 and is currently marked as 'Open'.
The case is being handled by Officer Jerry Mouse, ranked 'Constable', with badge number 'TX1006'.
The assigned officer is from Delhi Police, which has jurisdiction over Delhi, India.
The case is associated with an incident that occurred on 2025-03-30 at Kolkata, India.
The incident is classified as 'Burglary' and is currently 'Open'.
The victim in the case is Deepika Padukone, a female born on 1986-01-05.
The suspect is Selena Gomez, a female born on 1992-07-22.
A total of 1 piece(s) of evidence have been collected and documented.
Incident Description: Alia Bhatt's house was broken into.

==================================
```

- Listing cases

```
--- All Registered Cases ---


Case ID          : 1
Title            : SRK Robbery Case
Status           : Open
Open Date        : 2024-01-06
Close Date       : None
Assigned Officer : 1
Incident ID      : 1
----------------------------

Case ID          : 2
Title            : Aamir Homicide
Status           : Closed
Open Date        : 2025-02-11
Close Date       : 2025-03-01
Assigned Officer : 2
Incident ID      : 2
----------------------------

Case ID          : 3
Title            : Deepika Theft Case
Status           : Open
Open Date        : 2024-03-16
Close Date       : None
Assigned Officer : 3
Incident ID      : 3
```

FUTURE ENHANCEMENTS

In the future, this Crime Analysis and Reporting System can be enhanced with several features to improve functionality, usability, and scalability. One potential enhancement is the integration of a graphical user interface (GUI) or a web-based interface to make the system more user-friendly and accessible to non-technical users. Adding user authentication and role-based access control can ensure data privacy and restrict access based on user privileges such as admin, police officers, or analysts.

Moreover, the system can be extended to include geographical data mapping, using tools like Google Maps API, to visualize crime incidents based on location for better analysis and decision-making. Integration with machine learning algorithms can enable predictive analytics, helping law enforcement agencies identify crime hotspots and patterns.

Another valuable enhancement would be the implementation of notification and alert mechanisms, such as SMS or email notifications for critical incidents or case updates. Additionally, the project can be expanded to support real-time data updates and connections with external systems like police databases or emergency services for seamless information sharing.

Finally, incorporating a report dashboard with visual analytics using tools like Power BI or Tableau can provide deeper insights into crime trends, helping authorities take proactive measures. These enhancements would make the system more robust, efficient, and applicable in real-world crime analysis and reporting environments.

CONCLUSION

The **Crime Analysis and Reporting System** project has been successfully developed using a modular and layered architecture, ensuring scalability, maintainability, and real-world applicability. The system is designed to manage and analyze criminal activities efficiently, allowing users to register incidents, update statuses, manage cases, and generate reports.

The core business logic is defined through the interface in icrime_analysis_service.py, which outlines all essential functionalities like incident creation, status updates, date-wise searches, case generation, and report generation. This interface is implemented in crime_analysis_service_impl.py, where all the logic is built using robust coding practices and proper database handling.

The entity classes — incident.py, case.py, and status.py — encapsulate the data with private variables, constructors, getters, and setters, promoting data abstraction and reusability. These classes form the backbone of the system, representing real-world objects in a digital format.

The application connects to a SQL database using db_connection.py, which handles the database connection. It relies on property_util.py to read credentials and configuration from a properties file, keeping the application secure and configuration-based. This approach ensures flexibility and protects sensitive data like usernames and passwords.

To handle exceptions effectively and maintain system stability, a custom exception class — incident_number_not_found_exception.py — is created. This is thrown when an invalid or non-existent incident number is accessed. This makes the system more user-friendly and reliable.

All system functionalities are triggered from main_module.py, which acts as the entry point to the application. This script is responsible for interacting with the user and calling the appropriate service methods to perform the desired operations.

Finally, the entire application was validated through unit testing to ensure that key functionalities like incident creation and status updates perform accurately. Proper test cases were written and executed to verify correctness and robustness under different input scenarios.

In conclusion, this project integrates clean object-oriented programming, SQL-based database interaction, exception handling, and modular design to create a powerful and practical Crime Analysis and Reporting System. With potential future enhancements like a graphical user interface or web-based dashboard, this system can evolve into a full-fledged crime data management platform.