

Record Management System

Project Report

Group 3

Gudipalli Sri Sai Prabhath Reddy

Yashwant Dontam

857-398-7947 (Prabhath)

857-230-5413 (Yashwant)

gudipalli.s@northeastern.edu

dontam.y@northeastern.edu

Percentage of Effort Contributed by Student1: 50%

Percentage of Effort Contributed by Student2: 50%

Signature of Student 1: Gudipalli Sri Sai Prabhath Reddy

Signature of Student 2: Yashwant Dontam

Submission Date: 12/10/2023

1. Introduction

Data-driven policing could transform the way law enforcement agencies manage their data. A good Records Management System (RMS) would go a long way toward achieving notable efficiency and precision. A record management system can offer data entry, retrieval, and analysis. Correct and up-to-date information on the suspects, their criminal records, the case details, and the evidence items must be stored in the record management system. This could help identify suspects, track criminal records, and build case files. An RMS can also be used to store information within the organization, like details and schedules of the personnel working. This approach signifies a significant step ahead in adapting law enforcement to the demands of the 21st century. Down the road, this can help law enforcement identify trends, patterns, and potential leads, enhancing their ability to prevent and solve crimes efficiently and effectively. Our project takes a law enforcement division standpoint to manage and analyze data related to criminal activities, personnel, resources, and community interactions in their effort to enforce the law.

Theory for Record Management:

In our context of a law enforcement division's record management system, data can be recorded and managed from two fundamental perspectives: Case and Incident record management and Administrative record management. In case and incident management, the LED records core information about incident reports, cases, victims, suspects, and evidence. For every incident, the LED must record the incident ID, date and time, location, and status (open, closed, or under investigation). The case ID, assigned officers, incident, case description, and case status (open, closed, or pending) will be recorded for the cases. For the victim data, the victim ID, name, contact, and statement will be registered. The suspect's ID, name, date of birth, address, and criminal record must be stored. The data about the evidence, like evidence ID, type (physical, digital), description, chain of custody, and storage location will be recorded.

When it comes to personnel and administrative records management, the LED looks to record the officers' details, shift schedules, access control, and vehicle records. For the officers' data, officer ID, name, rank, badge number, and contact will be recorded. For every day of the week, the LED must record and store the schedule ID, employee ID, shift dates, and times. The user IDs, passwords, and role-based access should be stored for authentication and access control. For the vehicles owned by the department, vehicle ID, model, license plate number, assigned officer, and maintenance history will be recorded.

Other requirements:

- 1) One incident report can be connected to 0 to an infinite number of cases; one case should have one incident report.
- 2) Law enforcement officers can be associated with multiple cases, but each case must have at least one appointed officer.
- 3) Cases can be linked with 0 to multiple number of evidence items.
- 4) Each suspect must be linked with at least one case, but cases should have at least one suspect to multiple suspects.
- 5) Every case should include information about at least one victim, while the victim records can be disconnected to zero to infinite cases.
- 6) Every case should have a record of at least one administrative document, while the administrative documents can be related to infinite cases.
- 7) User accounts control access to the system, while each user has specific access rights

2. Methodology

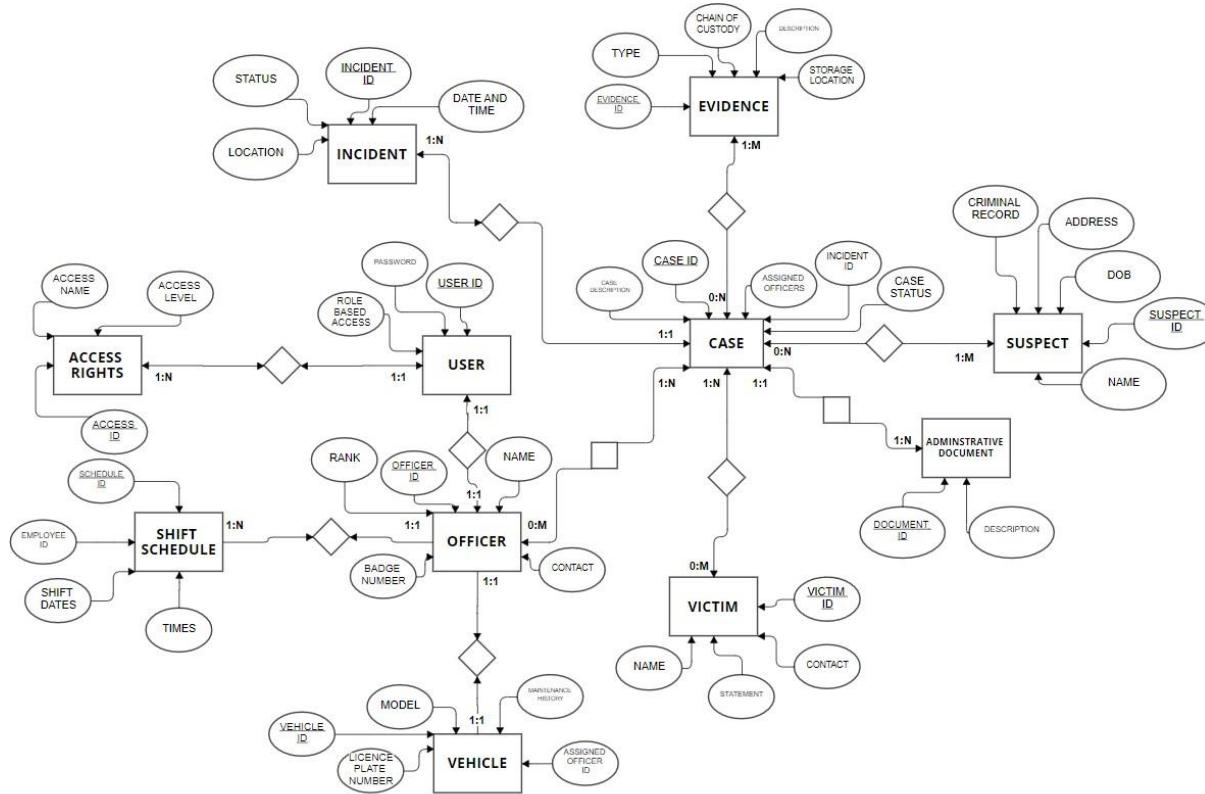


Figure 2.1: EER Model

The EER diagram models a law enforcement or security system, detailing how incidents, cases, evidence, suspects, officers, victims, and vehicles are interrelated. Key points include:

- Incidents are recorded with time and location details, and can be linked to multiple pieces of evidence and cases.
- Evidence is cataloged with unique identifiers and has a tracking mechanism for custody and storage.
- Cases are built around incidents and can involve multiple suspects and officers, with each case having a unique status.
- Suspects and victims have personal details recorded, and suspects additionally have criminal records linked.
- Officers have ranks, badge numbers, and are assigned to shifts and vehicles, with their schedules and vehicle maintenance histories tracked.
- Administrative documents are maintained for supplementary case documentation. Users of the system have role-based access, indicating a secure, permission-based system.
- Access rights are determined by levels and names, controlling what parts of the system users can interact with

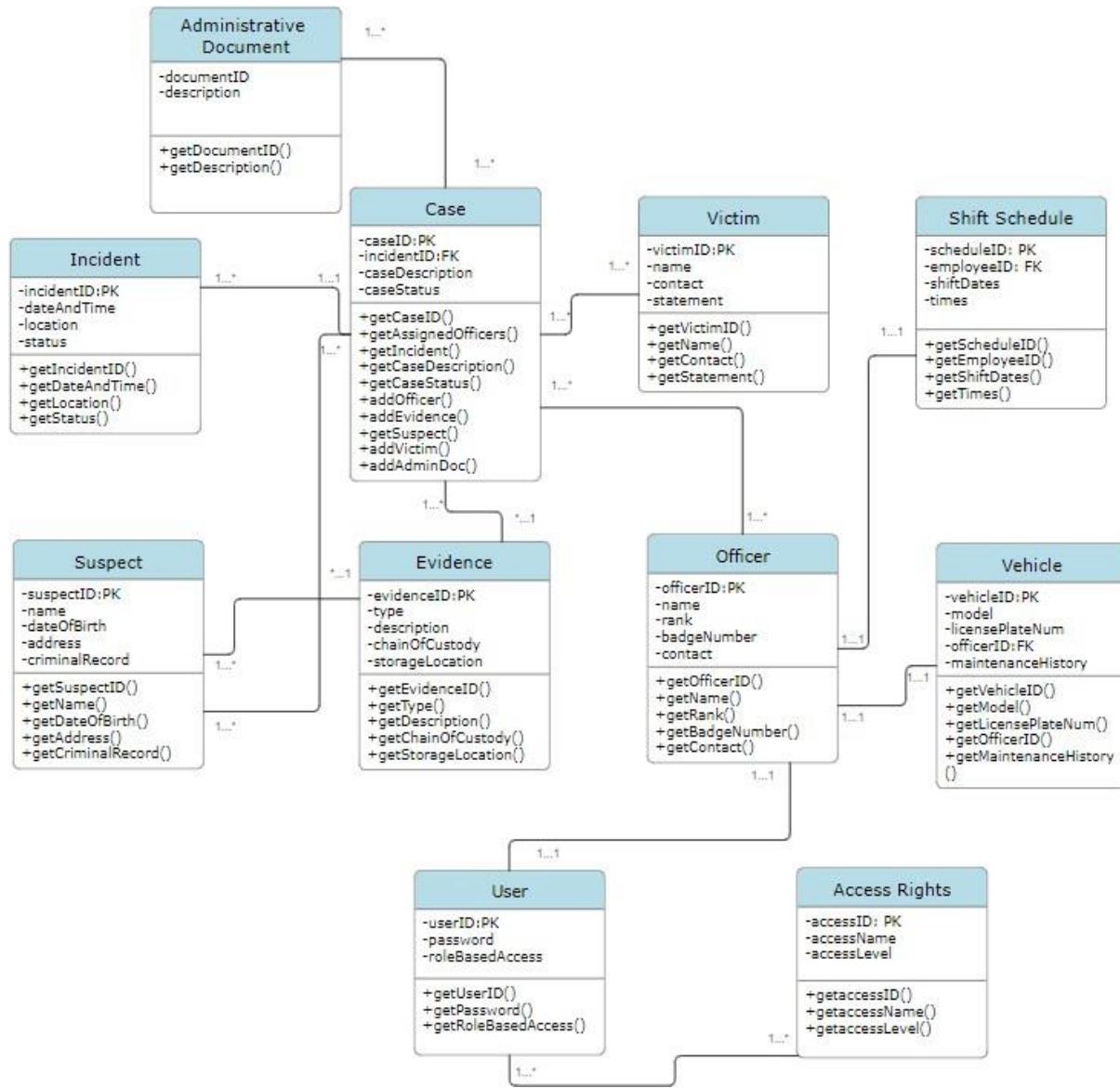


Figure 2.2: UML Model

The UML class diagram models the structure of a system likely used by law enforcement to manage incidents, cases, and related information. It outlines the attributes and methods of entities such as incidents, cases, suspects, evidence, officers, vehicles, victims, users, access rights, and administrative documents, specifying how these entities are related and interact with each other. Each class has attributes unique to the entity it represents, as well as methods to retrieve or manipulate these attributes. Relationships between classes indicate that an incident can be linked to multiple cases, suspects, and pieces of evidence, while officers are associated with vehicles and shift schedules. Users have role-based access, suggesting a system with controlled permissions. The structure implies a complex, interconnected system with defined operations to manage and access law enforcement data.

3. Relational Model

Case (CaseID, Case Description, Assigned Officers, Case Status)
Primary key: CaseID

Involved (CaseID, SuspectID)

Foreign keys: CaseID, SuspectID ; NOT NULL

CaseID referring to the primary key in the Entity Case;

SuspectID referring to the primary key in the Entity Suspect.

Suspect (SuspectID, SuspectName, SuspectDOB, SuspectAddress, CriminalRecord)
Primary key: SuspectID

Administrative Document (DocumentID, Description, CaseID)

Primary key: DocumentID;

Foreign key: CaseID; NOT NULL

CaseID referring to the primary key in the Entity Case

Victim (VictimID, VictimName, VictimContact, VictimStatement, CaseID)

Primary key: VictimID;

Foreign key: CaseID; NOT NULL

CaseID referring to the primary key in the Entity Case

Evidence (EvidenceID, EvidenceType, ChainOfCustody, EvidenceDescription, StorageLocation, CaseID)

Primary key: EvidenceID;

Foreign key: CaseID; NOT NULL

CaseID referring to the primary key in the Entity Case

Incident (IncidentID, IncidentLocation, IncidentStatus, IncidentDate, IncidentTime, CaseID)

Primary key: IncidentID;

Foreign key: CaseID; NOT NULL

CaseID referring to the primary key in the Entity Case

AssignedTo (CaseID, OfficerID)

Foreign keys: CaseID, OfficerID; NOT NULL

CaseID referring to the primary key in the Entity Case;

OfficerID referring to the primary key in the Entity Officer.

Officer (OfficerID, OfficerName, OfficerContact, OfficerRank, BadgeNumber, VehicleID, ScheduleID, UserID)

Primary key: OfficerID;

Foreign keys: VehicleID, ScheduleID, UserID; NOT NULL

VehicleID referring to the primary key in the Entity Vehicle;

ScheduleID referring to the primary key in the Entity Schedule;

UserID referring to the primary key in the Entity User.

User (UserID, RoleBasedAccess, Password)

Primary key: UserID

Access Rights (AccessID, AccessName, AccessLevel, UserID)

Primary key: AccessID;

Foreign keys: UserID; NOT NULL

UserID referring to the primary key in the Entity User

Vehicle (VehicleID, VehicleModel, LicencePlateNumber, MaintenanceHistory)

Primary key: VehicleID

Shift Schedule (ScheduleID, ShiftDates, ShiftTime)

Primary key: ScheduleID

4. SQL Demonstration

Queries showing the Tables and the Columns

1.

CREATE TABLE

```
AccessRights ( AccessID  
INT PRIMARY KEY,  
AccessName TEXT,  
AccessLevel TEXT,  
UserID INT NOT  
NULL,  
FOREIGN KEY (UserID) REFERENCES User(UserID)  
);
```

The screenshot shows a database management system (DBMS) interface. The top navigation bar includes icons for file operations, a search bar, and a toolbar with various tools. The left sidebar displays the 'Navigator' with 'SCHEMAS' expanded to show 'classicmodels', 'purchase', 'rms', and 'rms1'. Under 'rms1', 'Tables' are listed: 'accessrights', 'administrativedocume', 'assignedto', 'cases', 'evidences', 'incidents', 'involved', 'officers', 'shiftschedule', 'suspects', 'user', 'vehicles', and 'victims'. Below the tables are 'Views' and 'Stored Procedures'. The bottom left shows 'Administration' and 'Schemas' tabs, with 'Information' at the bottom. The main area shows the 'accessrights' table selected. The table has columns: AccessID, AccessName, AccessLevel, and UserID. The data grid shows the following rows:

AccessID	AccessName	AccessLevel	UserID
5000	ManageCases	Full	4500
5001	ManageUsers	Read-Only	4501
5002	ManageUsers	NoAccess	4502
5003	ViewCases	Read-Only	4503
5004	ViewReports	NoAccess	4504
5005	ManageUsers	Limited	4505
5006	ManageCases	Read-Only	4506
5007	ViewReports	Read Only	4507

The bottom right pane shows the 'Output' section with two entries:

- # 41 22:59:31 SELECT * FROM suspects WHERE CriminalRecord = 'Yes' LIMIT 0, 1000
- # 42 23:00:37 SELECT * FROM suspects WHERE CriminalRecord = 'Yes' LIMIT 0, 1000

Figure 4.1.1

2.

CREATE TABLE

```
AdministrativeDocuments (
    DocumentID INT PRIMARY KEY,
    Description TEXT,
    CaseID INT NOT NULL,
    FOREIGN KEY (CaseID) REFERENCES Cases(CaseID)
);
```

The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'Schemas' tree, with 'rms1' selected. Under 'Tables', the 'administrative documents' table is highlighted. The main area shows the 'Result Grid' with the following data:

DocumentID	Description	CaseID
1000	mauris sapien, cursus in, hendrerit consetetur...	1234
1001	eu metus. In lorem. Donec elementum, lorem ut...	1235
1002	mauris blandit mattis. Cras eget nisi dictum aug...	1236
1003	ultrices iaculis odio. Nam interdum enim non nisi...	1237
1004	nec, diam. Duis mi enim, condimentum eget, vol...	1238
1005	pretium aliquet, metus urna convallis erat, eget ...	1239
1006	eu tempor erat neque non quam. Pellentesque ...	1240
1007	mauris non ante bibendum ullamcorper	1241

Below the grid, the 'Table: administrative documents' section shows the table structure:

Columns:
DocumentID int PK
Description text
CaseID int

The 'Output' pane at the bottom shows the history of actions:

#	Time	Action
42	23:00:37	SELECT * FROM suspects WHERE CriminalRecord = 'Yes' LIMIT 0, 1000
43	23:03:10	SELECT * FROM rms1.accessrights LIMIT 0, 1000

Figure 4.1.2

3.

CREATE TABLE

```
AssignedTo (CaseID INT  
NOT NULL, OfficerID  
INT NOT NULL,  
PRIMARY KEY (CaseID, OfficerID),  
FOREIGN KEY (CaseID) REFERENCES Cases(CaseID),  
FOREIGN KEY (OfficerID) REFERENCES Officers(OfficerID)  
);
```

The screenshot shows a database management interface with the following details:

- Schemas:** classicmodels, purchase, rms, rms1 (selected).
- Tables:** assignedto (selected), accessrights, administrative documents, cases, evidences, incidents, involved, officers, shiftschedule, suspects, user, vehicles, victims.
- Views:** None.
- Stored Procedures:** None.
- Query Bar:** SELECT * FROM rms1.assignedto;
- Result Grid:** Shows data for the assignedto table.

CaseID	OfficerID
1234	3000
1235	3001
1236	3002
1237	3003
1238	3004
1239	3005
1240	3006
1241	3007

- Information:** Table: assignedto, Columns: CaseID int, OfficerID int.
- Output:** Action Output, showing a log entry: # 43 Time 23:03:10 Action SELECT * FROM rms1.accessrights LIMIT 0, 1000.

Figure 4.1.3

4.

CREATE TABLE cases (

CaseID INT PRIMARY

KEY,

CaseDescription

TEXT,

AssignedOfficers

TEXT, CaseStatus

TEXT

);

The screenshot shows a database interface with the following details:

- Schemas:** rms1
- Tables:** cases
- Columns:**

Column	Type
CaseID	int PK
CaseDescription	text
AssignedOfficer	text
CaseStatus	text
- Data Grid:**

CaseID	CaseDescription	AssignedOfficer	CaseStatus
1234	nec ligula consectetur rhoncus. Nullam velit dui...	Nathan Haley	Closed
1235	massa. Quisque porttitor eros	Megan Casey	Opened
1236	ornare. In faucibus. Morbi vehicula. Pellentesqu...	Leila Baker	Opened
1237	Suspendisse commodo tincidunt nibh. Phasellus ...	Adele Drake	Closed
1238	erat vitae risus. Duis	Wing Fulton	Pending
1239	Sed malesuada augue ut lacus. Nulla tincidunt, ...	Phyllis Ford	Opened
1240	velit. Aliquam nisl. Nulla eu neque pellentesque ...	Xenos Pratt	Closed
1241	malesuada augue ut lacus. Nulla tincidunt, ...	Caroline Warden	Closed
- Action Output:**

#	Time	Action	Rows
44	23:03:34	SELECT * FROM rms1.'administrative documents' LIMIT 0, 1000	501
45	23:03:39	SELECT * FROM rms1.'administrative documents' LIMIT 0, 1000	501

Figure 4.1.4

5.

CREATE TABLE evidences (

 EvidenceID INT PRIMARY

 KEY,

 EvidenceType TEXT,

 ChainOfCustody TEXT,

 EvidenceDescription

 TEXT, StorageLocation

 TEXT, CaseID INT NOT

 NULL,

 FOREIGN KEY (CaseID) REFERENCES Case(CaseID)

);

Table: evidences

Columns:

EvidenceID	int	PK
EvidenceType	text	
ChainOfCustody	text	
EvidenceDescription	text	
StorageLocation	text	
CaseID	int	

Action Output

#	Time	Action	Message
45	23:03:39	SELECT * FROM rms1.'administrative documents' LIMIT 0, 1000	500 row(s) retu
46	23:05:15	SELECT * FROM rms1.accessrights LIMIT 0, 1000	500 row(s) retu
47	23:05:45	SELECT * FROM rms1.'administrative documents' LIMIT 0, 1000	500 row(s) retu
48	23:06:18	SELECT * FROM rms1.assignedto LIMIT 0, 1000	500 row(s) retu

Figure 4.1.5

6.

```
CREATE TABLE incidents (
    IncidentID INT PRIMARY
    KEY,
    IncidentLocation
    TEXT, IncidentStatus
    TEXT, IncidentDate
    DATE, IncidentTime
    TIME, CaseID INT
    NOT NULL,
    FOREIGN KEY (CaseID) REFERENCES Cases(CaseID)
);
```

The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'Schemas' tree, with 'rms1' selected. Under 'Tables', the 'incidents' table is highlighted. The main area shows the 'Result Grid' with the following data:

IncidentID	IncidentLocation	IncidentStatus	IncidentDate	IncidentTime	CaseID
2500	Iowa	Closed	Jan 24, 2023	3:29 AM	1234
2501	Idaho	Resolved	Feb 8, 2023	8:40 AM	1235
2502	Kentucky	UnderInvestigation	Nov 22, 2022	8:16 AM	1236
2503	Iowa	Closed	Dec 13, 2023	2:41 PM	1237
2504	Wyoming	Resolved	Mar 18, 2024	10:30 PM	1238
2505	Wyoming	Resolved	Aug 16, 2023	2:14 AM	1239
2506	Pennsylvania	Closed	Oct 12, 2024	1:29 AM	1240
2507	Missouri	Closed	Oct 8, 2022	5:55 AM	1241

The 'Action Output' pane at the bottom lists recent database actions:

#	Time	Action	Message
46	23:05:15	SELECT * FROM rms1.accessrights LIMIT 0, 1000	500 r
47	23:05:45	SELECT * FROM rms1.administrative documents' LIMIT 0, 1000	500 r
48	23:06:18	SELECT * FROM rms1.assignedto LIMIT 0, 1000	500 r
49	23:07:55	SELECT * FROM rms1.cases LIMIT 0, 1000	500 r

Figure 4.1.6

7.

CREATE TABLE Involved

```
( CaseID INT NOT  
NULL, SuspectID INT  
NOT NULL,  
PRIMARY KEY (CaseID, SuspectID),  
FOREIGN KEY (CaseID) REFERENCES Cases(CaseID),  
FOREIGN KEY (SuspectID) REFERENCES Suspect(SuspectID)
```

);

The screenshot shows the MySQL Workbench interface. On the left, the Navigator pane displays the 'schemas' section with 'rms1' selected. Under 'rms1', the 'Tables' section lists various tables like 'accessrights', 'cases', 'evidences', etc., with 'involved' being the currently selected table. The main workspace shows the 'involved' table's structure and data. The table has two columns: 'CaseID' and 'SuspectID'. The data grid displays the following 10 rows:

CaseID	SuspectID
1234	1740
1235	1741
1236	1742
1237	1743
1238	1744
1239	1745
1240	1746
1241	1747
1242	1748

At the bottom of the interface, the 'Output' pane shows the SQL query used to retrieve the data: 'SELECT * FROM rms1.involved;' and the message '500 row(s) returned'.

Figure 4.1.7

8.

```
CREATE TABLE Officers (
    OfficerID INT PRIMARY
    KEY,
    OfficerName TEXT,
    OfficerContact TEXT,
    OfficerRank TEXT,
    BadgeNumber TEXT,
    VehicleID INT NOT
    NULL, ScheduleID INT
    NOT NULL, UserID INT
    NOT NULL,
    FOREIGN KEY (VehicleID) REFERENCES Vehicle(VehicleID),
    FOREIGN KEY (ScheduleID) REFERENCES
    Schedule(ScheduleID), FOREIGN KEY (UserID) REFERENCES
    User(UserID)
);
```

The screenshot shows a database interface with the following details:

- Schemas:** rms1
- Tables:** officers
- Result Grid:**

OfficerID	OfficerName	OfficerContact	OfficerRank	BadgeNumber	VehicleID	ScheduleID	UserID
3000	Gisela Farley	(275) 462-3737	Lieutenant	GPC54FVK10W	3500	4000	4500
3001	Hiroko Burt	(207) 265-9587	Sergeant	MQW68ZU19FU	3501	4001	4501
3002	Gregory Townsend	(370) 638-4515	Inspector	DJB66P0O2BT	3502	4002	4502
3003	Sonya Hayes	(585) 262-9722	Chief	QD122B1Y9MZ	3503	4003	4503
3004	Dominic Greer	(813) 738-8474	Inspector	KF122PMQ7GX	3504	4004	4504
3005	Francesca Keller	(472) 985-6378	Captain	OBP90JWP2PF	3505	4005	4505
3006	Tyler Singleton	(636) 172-9602	Lieutenant	AYF34CUB3QI	3506	4006	4506
3007	Samuel Parker	(662) 477-3077	Corporal	MEE127M1PNU	3507	4007	4507
- Output:**

#	Time	Action	Message
48	23:06:18	SELECT * FROM rms1.assignedto LIMIT 0, 1000	500 row(s) returned
49	23:07:55	SELECT * FROM rms1.cases LIMIT 0, 1000	500 row(s) returned
50	23:08:22	SELECT * FROM rms1.evidences LIMIT 0, 1000	500 row(s) returned
51	23:08:47	SELECT * FROM rms1.incidents LIMIT 0, 1000	500 row(s) returned
52	23:09:25	SELECT * FROM rms1.involved LIMIT 0, 1000	500 row(s) returned

Figure 4.1.8

9.

CREATE TABLE

```
ShiftSchedule ( ScheduleID  
INT PRIMARY KEY,  
ShiftDates  
DATE,  
ShiftTime TIME  
);
```

The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'Schemas' tree, with 'rms1' selected. Under 'Tables', the 'shiftschedule' table is listed. The main pane shows the table structure and data. The 'Result Grid' shows the following data:

ScheduleID	ShiftDates	ShiftTime
4000	Sat, Nov 25th, 2023	2:13 PM
4001	Sat, Apr 22nd, 2023	1:29 AM
4002	Sat, Sep 16th, 2023	3:33 AM
4003	Wed, Jul 3rd, 2024	11:15 AM
4004	Tue, Jan 10th, 2023	8:45 PM
4005	Tue, Feb 28th, 2023	1:02 AM
4006	Thu, Jan 11th, 2024	6:39 AM
4007	Sat, Jun 7th, 2024	4:00 PM

The 'Output' tab at the bottom shows the query used to retrieve the data: 'SELECT * FROM rms1.shiftschedule;'. The message indicates '500 row(s) returned'.

Figure 4.1.9

10.

```
CREATE TABLE suspects (
    SuspectID INT PRIMARY
    KEY,
    SuspectName TEXT,
    SuspectDOB DATE,
    SuspectAddress
    TEXT,
    CriminalRecord
    TEXT
);
```

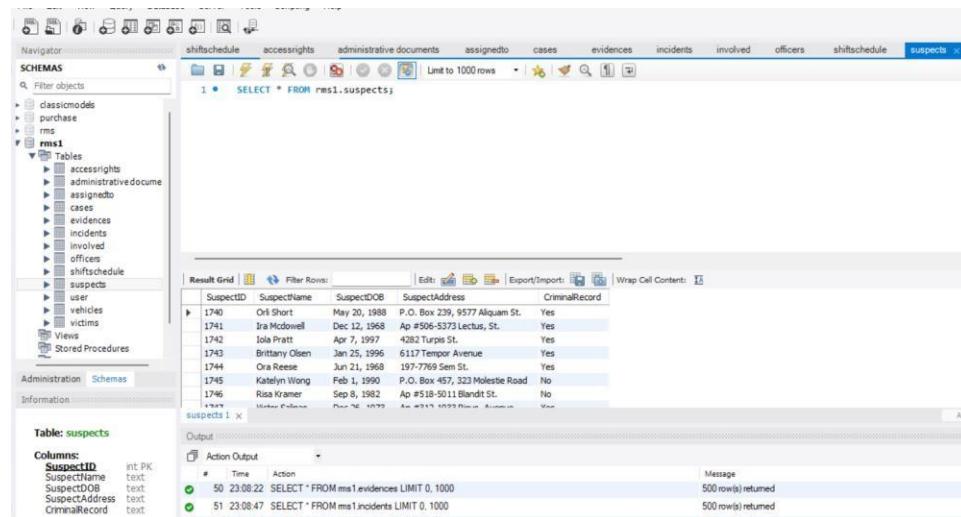


Figure 4.1.10

11.

```
CREATE TABLE Users (
    UserID INT PRIMARY
    KEY,
```

RoleBasedAccess

TEXT,Password

TEXT

);

Table: user

Columns:

UserID	int PK
RoleBasedAccess	text
Password	text

Result Grid:

User ID	Role Based Access	Password
4500	Analyst	VCT2D9nQ9O
4501	Officer	3B713ZV16J
4502	Officer	V3J02.YT+K2
4503	Analyst	NOT4QEh0F1
4504	Analyst	SDA1ST07720
4505	Officer	Z5Z56WV9K7S
4506	Guest	BKd86YTR2LT

Action Output:

- 51 23:08:47 SELECT * FROM rms1.incidents LIMIT 0, 1000

Message: 500 row(s) returned

Figure 4.1.11

12.

CREATE TABLE Vehicles (

 VehicleID INT PRIMARY

 KEY,

 VehicleModel TEXT,

 LicencePlateNumber

 TEXT,

 MaintenanceHistory

 TEXT

);

Table: vehicles

Columns:

VehicleID	int PK
VehicleModel	text
LicencePlateNumber	text
MaintenanceHistory	text

Result Grid:

VehicleID	VehicleModel	LicencePlateNumber	MaintenanceHistory
3500	Van	DH424WV2G	OilChange
3501	SUV	IPH03TVQ2N5	OilChange
3502	Van	OCC17VCF8U	EngineTune-Up
3503	Motorcycle	KH72WVQ19H	EngineTune-Up
3504	Motorcycle	DPK1STLH4SD	BatterReplacement
3505	Motorcycle	MQD225K93K0	BrakeInspection
3506	SUV	GWF21MJK4N	OilChange
3507	Van	SPY4460VV4H	BrakeInspection

Action Output:

- 52 23:09:25 SELECT * FROM rms1.involved LIMIT 0, 1000
- 53 23:10:00 SELECT * FROM rms1.officers LIMIT 0, 1000
- 54 23:10:25 SELECT * FROM rms1.shiftschedule LIMIT 0, 1000

Message: 500 row(s) returned

Message: 500 row(s) returned

Message: 500 row(s) returned

Figure 4.1.12

13.

```
CREATE TABLE Victims (
    VictimID INT PRIMARY KEY,
    VictimName TEXT,
    VictimContact TEXT,
    VictimStatement TEXT,
    CaseID INT NOT NULL,
    FOREIGN KEY (CaseID) REFERENCES Cases(CaseID)
);
```

The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'schemas' tree, with 'rms1' selected. The main area shows the 'victims' table structure. The table has the following columns:

VictimID	VictimName	VictimContact	VictimStatement	CaseID
1500	Henry Butler	(212) 450-4559	metus. Vivamus estmod urna. Nullam lobortis q...	1234
1501	Axel Finley	(503) 545-1784	magis nec quam. Curabitur vel lectus. Cum	1235
1502	Hannah Lawrence	(967) 997-7598	commodo auctor velit. Aliquam nisl. Nulla eu neq...	1236
1503	Zachary McDonald	(219) 131-8126	id enim. Curabitur massa. Vestibulum accumsan ...	1237
1504	Anal Luna	(418) 621-4850	a sollicitudin orci sem eget massa. Suspendisse ...	1238
1505	Jasper Cantu	(114) 306-1965	lula consetetur rhoncus. Nullam velit dui, se...	1239
1506	Driscoll Mcgee	(502) 700-1515	eget metus. In nec orci. Donec nibh. Quisque n...	1240

Below the table, the 'Output' pane shows the following history:

#	Time	Action	Message
53	23.10.25	SELECT * FROM rms1.officers LIMIT 0, 1000	500 row(s) returned
54	23.10.25	SELECT * FROM rms1.shiftschedule LIMIT 0, 1000	500 row(s) returned

Figure 4.1.13

4.2 Sample Queries

The screenshot shows the MySQL Workbench interface with the following details:

- File Bar:** File, Edit, View, Query, Database, Server, Tools, Scripting, Help.
- Navigator:** Local instance MySQL80, Schemas (classicmodels, purchase, rms, rms1), Tables (accessrights, administrativedocument, assignedto, cases, evidences, incidents, involved, officers, shiftschedule, suspects, user, vehicles, victims), Views, Stored Procedures, Functions.
- Query Editor:** Title: Query 1, Content: A multi-line SQL script for retrieving case details, and a note: "Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help." The script includes:


```

1 -- Retrieving details about cases, including CaseID, CaseDescription, AssignedOfficer, CaseStatus, and OfficerName
2 • SELECT cases.CaseID, cases.CaseDescription, cases.AssignedOfficer, cases.CaseStatus, officers.OfficerName
3   FROM cases
4     LEFT JOIN AssignedTo ON cases.CaseID = AssignedTo.CaseID
5     LEFT JOIN officers ON assignedto.OfficerID = officers.OfficerID;
6
7
8
9
      
```
- Result Grid:** Shows the output of the query with columns: CaseID, CaseDescription, AssignedOfficer, CaseStatus, and OfficerName. The data is as follows:

CaseID	CaseDescription	AssignedOfficer	CaseStatus	OfficerName
1234	nec ligula consectetur rhoncus. Nullam velit dui...	Nathan Hale	Closed	Gisela Farley
1235	massa. Quisque porttitor eros...	Megan Casey	Open	Hiroko Burt
1236	ornare. In faucibus. Morbi vehicula. Pellentesqu...	Lela Baker	Open	Gregory Townsend
1237	Suspendisse commodo tincidunt nibh. Phasellus ...	Adele Drake	Closed	Sonya Hayes
1238	erat vitae risus. Duis...	Wing Fulton	Pending	Dominic Greer
1239	Sed malesuada augue ut lacus. Nulla tincidunt, ...	Phyllis Ford	Opened	Francesca Keller
1240	velit. Aliquam nisl. Nulla eu neque pellentesque ...	Xenos Pratt	Closed	Tyler Singleton
1241	mollis...	Sophia Hardin	Closed	Samantha Burton
- Action Output:** Shows a log of actions and their results:

#	Time	Action	Message	Duration / Fetch
34	22:45:12	Apply changes to accessrights	Changes applied	0.000 sec / 0.000 sec
35	22:45:23	SELECT AccessName, AccessLevel FROM accessrights WHERE UserID = 4513 LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec
36	22:45:57	SELECT AccessName, AccessLevel FROM accessrights WHERE UserID = '4513', '4678'	Error Code: 1064. You have an error in your SQL syntax; check the manual that corresponds to your MySQL se...	0.000 sec
37	22:46:10	SELECT AccessName, AccessLevel FROM accessrights WHERE UserID = '4513' LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec
38	22:47:02	SELECT AccessName, AccessLevel FROM accessrights WHERE UserID IN (4513, 4890, 4657) LIMIT 0, 1...	3 row(s) returned	0.000 sec / 0.000 sec
39	22:49:46	SELECT cases.CaseID, cases.CaseDescription, cases.AssignedOfficer, cases.CaseStatus, officers.OfficerName...	500 row(s) returned	0.000 sec / 0.000 sec

Figure 4.2.1

Schemas

```

1 -- Retrieving information about incidents, specifically their location, date, and time, for incidents that occurred after January 1, 2022
2 • SELECT IncidentLocation, IncidentDate, IncidentTime
3 FROM incidents
4 WHERE IncidentDate > '2022-01-01'
5 LIMIT 0, 1000
6

```

Result Grid

IncidentLocation	IncidentDate	IncidentTime
Iowa	Jan 24, 2023	3:29 AM
Idaho	Feb 8, 2023	8:40 AM
Kentucky	Nov 22, 2022	8:16 AM
Iowa	Dec 13, 2023	2:41 PM
Wyoming	Mar 18, 2024	10:30 PM
Wyoming	Aug 16, 2023	2:14 AM
Pennsylvania	Oct 12, 2024	1:29 AM
Missouri	Oct 8, 2022	5:55 AM

Output

#	Time	Action	Message
35	22:45:23	SELECT AccessName, AccessLevel FROM accessrights WHERE UserID = 4513 LIMIT 0, 1000	1 row(s) returned
36	22:45:52	SELECT AccessName, AccessLevel FROM accessrights WHERE UserID = '4513', '4678'	Error Code: 1064. You have an error in your SQL syntax; check the manual that corresponds
37	22:46:10	SELECT AccessName, AccessLevel FROM accessrights WHERE UserID = '4513' LIMIT 0, 1000	1 row(s) returned
38	22:47:02	SELECT AccessName, AccessLevel FROM accessrights WHERE UserID IN ('4513', '4890', '4657') LIMIT 0, 1...	3 row(s) returned
39	22:49:46	SELECT cases.CaseID, cases.CaseDescription, cases.AssignedOfficer, cases.CaseStatus, officers.OfficerNam...	500 row(s) returned
40	22:56:55	SELECT IncidentLocation, IncidentDate, IncidentTime FROM incidents WHERE IncidentDate > '2022-01-01'...	500 row(s) returned

Figure 4.2.2

File Edit View Query Database Server Tools Scripting Help

Navigator: shiftschedule

Schemas

- classicmodels
- purchase
- rms
- rms1**
 - Tables
 - accessrights
 - administrativedocument
 - assignedto
 - cases
 - evidences
 - incidents
 - involved
 - officers
 - shiftschedule
 - suspects
 - user
 - vehicles
 - victims
- Views
- Stored Procedures

Administration Schemas

Information

Table: **accessrights**

Columns:

AccessID	int PK		
AccessName	text		
AccessLevel	text		
UserID	int		
#	Time	Action	Message
37	22:46:10	SELECT AccessName, AccessLevel FROM accessrights WHERE UserID = '4513' LIMIT 0, 1000	1 row(s) returned
38	22:47:02	SELECT AccessName, AccessLevel FROM accessrights WHERE UserID IN ('4513', '4890', '4657') LIMIT 0, 1...	3 row(s) returned
39	22:49:46	SELECT cases.CaseID, cases.CaseDescription, cases.AssignedOfficer, cases.CaseStatus, officers.OfficerNam...	500 row(s) returned
40	22:56:55	SELECT IncidentLocation, IncidentDate, IncidentTime FROM incidents WHERE IncidentDate > '2022-01-01'...	500 row(s) returned
41	22:59:31	SELECT * FROM suspects WHERE CriminalRecord = 'Yes' LIMIT 0, 1000	257 row(s) returned
42	23:00:37	SELECT * FROM suspects WHERE CriminalRecord = 'Yes' LIMIT 0, 1000	257 row(s) returned

Figure 4.2.3

The screenshot shows a MySQL Workbench interface with the following details:

- Navigator:** Shows the database structure under the schema **rms1**, including tables like **accessrights**, **administrativedocument**, **assignedto**, **cases**, **evidences**, **incidents**, **involved**, **officers**, **shiftschedule**, **suspects**, **user**, **vehicles**, and **victims**.
- SQL Editor:** Displays the following SQL query:


```

1 -- Displaying all the values of UserID and Password columns
2 • SELECT UserID, Password
3 FROM user;
4
      
```
- Result Grid:** Shows the results of the query, listing UserID and Password for users 4500 through 4506. The results are:

UserID	Password
4500	VCY22HMX8PO
4501	IBY13ZWL6JE
4502	VJ00LYT4CZ
4503	NOT42EYN2FI
4504	SDA15TDY7ZO
4505	ZSZ56VWK9TS
4506	BKD86YTR2LT
4507	UQD9CTTMM
- Output:** Shows the history of actions taken during the session, including the execution of various queries against the **rms1** database.

Figure 4.2.4

Nested query:

The screenshot shows the MySQL Workbench interface with a query editor window open. The query being run is:

```
1 •  SELECT CaseID, CaseStatus
2   FROM cases
3 WHERE CaseID IN (SELECT CaseID FROM assignedto WHERE OfficerID = 123);
```

The results grid displays one row:

CaseID	CaseStatus
NULL	NULL

The status bar at the bottom right of the interface indicates: "Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help."

Below the main window, the "Output" pane shows the execution log:

#	Time	Action	Message	Duration / Fetch
1	20:30:30	SELECT CaseID, COUNT(*) AS NumberOfIncidents... 500 row(s) returned		0.015 sec / 0.000 sec
2	20:30:59	SELECT CaseID, CaseStatus FROM cases WHERE... 0 row(s) returned		0.016 sec / 0.000 sec

Figure 4.2.5

Correlated Query:

The screenshot shows the MySQL Workbench interface with a query editor and results grid.

Query Editor:

```
SELECT OfficerName,
       (SELECT COUNT(*)
        FROM assignedto
        WHERE assignedto.OfficerID = officers.OfficerID
       ) AS NumberOfCases
  FROM officers;
```

Result Grid:

OfficerName	NumberOfCases
Gisela Farley	1
Hiroko Burt	1
Gregory Townsend	1
Sonya Hayes	1
Dominic Greer	1
Francesca Keller	1
Tyler Singleton	1
Samantha Burton	1
Ferris Barrera	1
Lewis Bird	1
Uriel Peters	1
Cheryl Green	1
Andrew Darve	1

Action Output:

#	Time	Action	Message	Duration / Fetch
1	20:30:30	SELECT CaseID, COUNT(*) AS NumberOfIncidents...	500 row(s) returned	0.015 sec / 0.000 sec
2	20:30:59	SELECT CaseID, CaseStatus FROM cases WHERE...	0 row(s) returned	0.016 sec / 0.000 sec
3	20:32:35	SELECT OfficerName, (SELECT COUNT(*) F...	500 row(s) returned	0.016 sec / 0.000 sec

Figure 4.2.6

UNION (Set Operation):

The screenshot shows the MySQL Workbench interface with a query editor window. The query is:

```
1 ●  SELECT CaseID FROM cases
2   UNION
3  SELECT CaseID FROM incidents;
```

The results grid displays the following data:

CaseID
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246

The output pane shows the following log entries:

#	Time	Action	Message	Duration / Fetch
1	20:30:30	SELECT CaseID, COUNT(*) AS NumberOfIncidents...	500 row(s) returned	0.015 sec / 0.000 sec
2	20:30:59	SELECT CaseID, CaseStatus FROM cases WHERE...	0 row(s) returned	0.016 sec / 0.000 sec
3	20:32:35	SELECT OfficerName, (500 row(s) returned	0.016 sec / 0.000 sec
4	20:33:55	SELECT CaseID FROM cases WHERE CaseStatus...	116 row(s) returned	0.015 sec / 0.000 sec
5	20:34:08	SELECT CaseID FROM cases UNION SELECT Ca...	500 row(s) returned	0.000 sec / 0.000 sec

Figure 4.2.7

Subquery in FROM:

The screenshot shows the MySQL Workbench interface with a query editor window open. The query is:

```
1 ●  SELECT AVG(IncidentCount) AS AverageIncidentsPerCase
2   ○  FROM (
3       SELECT c.CaseID, COUNT(*) AS IncidentCount
4       FROM cases c
5       LEFT JOIN incidents i ON c.CaseID = i.CaseID
6       GROUP BY c.CaseID
7   ) AS CaseIncidentCounts;
```

The result grid shows a single row with the value 1.0000. A context help message on the right says: "Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help."

The bottom pane shows the execution history:

#	Time	Action	Message	Duration / Fetch
1	20:30:30	SELECT CaseID, COUNT(*) AS NumberOfIncidents...	500 row(s) returned	0.015 sec / 0.000 sec
2	20:30:59	SELECT CaseID, CaseStatus FROM cases WHERE...	0 row(s) returned	0.016 sec / 0.000 sec
3	20:32:35	SELECT OfficerName, (SELECT COUNT(*) F...	0.016 sec / 0.000 sec
4	20:33:55	SELECT CaseID FROM cases WHERE CaseStatus...	116 row(s) returned	0.015 sec / 0.000 sec
5	20:34:08	SELECT CaseID FROM cases UNION SELECT Ca...	500 row(s) returned	0.000 sec / 0.000 sec
6	20:35:50	SELECT AVG(IncidentCount) AS AverageIncidents...	1 row(s) returned	0.016 sec / 0.000 sec

Figure 4.2.8

NOT EXISTS:

The screenshot shows the MySQL Workbench interface with a query editor window. The query is:

```
1 ●  SELECT *
  2   FROM cases c
  3   WHERE NOT EXISTS (
  4     SELECT 1
  5       FROM assignedto a
  6      WHERE a.CaseID = c.CaseID
  7 );
```

The result grid shows no rows returned:

CaseID	CaseDescription	AssignedOfficer	CaseStatus
NULL	NULL	NULL	NULL

The context help panel on the right indicates: "Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help."

The session output pane shows the execution log:

#	Time	Action	Message	Duration / Fetch
2	20:30:59	SELECT CaseID, CaseStatus FROM cases WHERE ...	0 row(s) returned	0.016 sec / 0.000 sec
3	20:32:35	SELECT OfficerName, (SELECT COUNT(*) ...	500 row(s) returned	0.016 sec / 0.000 sec
4	20:33:55	SELECT CaseID FROM cases WHERE CaseStat...	116 row(s) returned	0.015 sec / 0.000 sec
5	20:34:08	SELECT CaseID FROM cases UNION SELECT C...	500 row(s) returned	0.000 sec / 0.000 sec
6	20:35:50	SELECT AVG(IncidentCount) AS AverageIncident...	1 row(s) returned	0.016 sec / 0.000 sec
7	20:37:10	SELECT * FROM cases c WHERE NOT EXISTS ...	0 row(s) returned	0.000 sec / 0.000 sec

Figure 4.2.9

5. Nosql Implementation

1. Listing all the cases where the case status is “Closed”

The screenshot shows the MongoDB Compass interface. The left sidebar lists databases and collections, with 'cases' selected under 'rms_mongodb'. The main pane displays the 'rms_mongodb.cases' collection, which contains 500 documents. A filter is applied to show documents where CaseStatus is "Closed". Four documents are listed:

- CaseID: 1234, CaseDescription: "nec ligula consetetur rhoncus. Nullam velit dui, semper et, lacinia ...", AssignedOfficer: "Nathan Haley", CaseStatus: "Closed"
- CaseID: 1237, CaseDescription: "Suspendisse commodo tincidunt nibh. Phasellus nulla.", AssignedOfficer: "Adele Drake", CaseStatus: "Closed"
- CaseID: 1240, CaseDescription: "velit. Aliquam nisl. Nulla eu neque pellentesque massa lobortis.", AssignedOfficer: "Xenos Pratt", CaseStatus: "Closed"
- CaseID: 1241, CaseDescription: "venenatis vel ...", AssignedOfficer: null, CaseStatus: "Closed"

2. Listing all incidents and their associated cases where the incident status is Reported

The screenshot shows the MongoDB Compass interface. The left sidebar lists databases and collections, with 'incidents' selected under 'local'. The main pane shows the results of an aggregation query run in the mongo shell:

```
> db.incidents.aggregate([
  { $match: { "IncidentStatus": "Reported" } },
  { $lookup: {
    from: "cases",
    localField: "CaseID",
    foreignField: "CaseID",
    as: "case_details"
  }}
])
< [
  {
    _id: ObjectId("656d0ed245dd8b9174ae320c"),
    IncidentID: 2515,
    IncidentLocation: 'Kentucky',
    IncidentStatus: 'Reported',
    IncidentDate: '27-Mar-23',
    IncidentTime: '3:37:00',
    CaseID: 1249,
    case_details: [
      {
        _id: ObjectId("656d0e3c45dd8b9174ae3017"),
        CaseID: 1249,
        CaseDescription: 'venenatis vel ...'
      }
    ]
  }
]
```

3. Listing all cases with their respective number of involved suspects

The screenshot shows the MongoDB Compass interface. The top bar displays "MongoDB Compass - localhost:27017/rms_mongodb.assignedto". The left sidebar shows "My Queries" and "Documents" for the "rms_mongodb.assignedto" collection. The main area shows the aggregation pipeline:

```
> db.cases.aggregate([
  { $lookup: {
    from: "involved",
    localField: "CaseID",
    foreignField: "CaseID",
    as: "suspects_involved"
  }},
  { $project: {
    "CaseDescription": 1,
    "NumberOfSuspects": { $size: "$suspects_involved" }
  }}
])
< [
  {
    _id: ObjectId("656d0e3c45dd8b9174ae3008"),
    CaseDescription: 'nec ligula consectetur rhoncus. Nullam velit dui, semper et, lacinia vitae, sodales at, velit. Pellentesque ultricies dignissim lacus. Aliquam',
    NumberOfSuspects: 1
  },
  {
    _id: ObjectId("656d0e3c45dd8b9174ae3009"),
    CaseDescription: 'massa. Quisque porttitor eros',
    NumberOfSuspects: 1
  }
]
```

The status bar at the bottom indicates "Cloudy" weather, "6°C", "ENG US", "9:40 PM", and the date "12/3/2023".

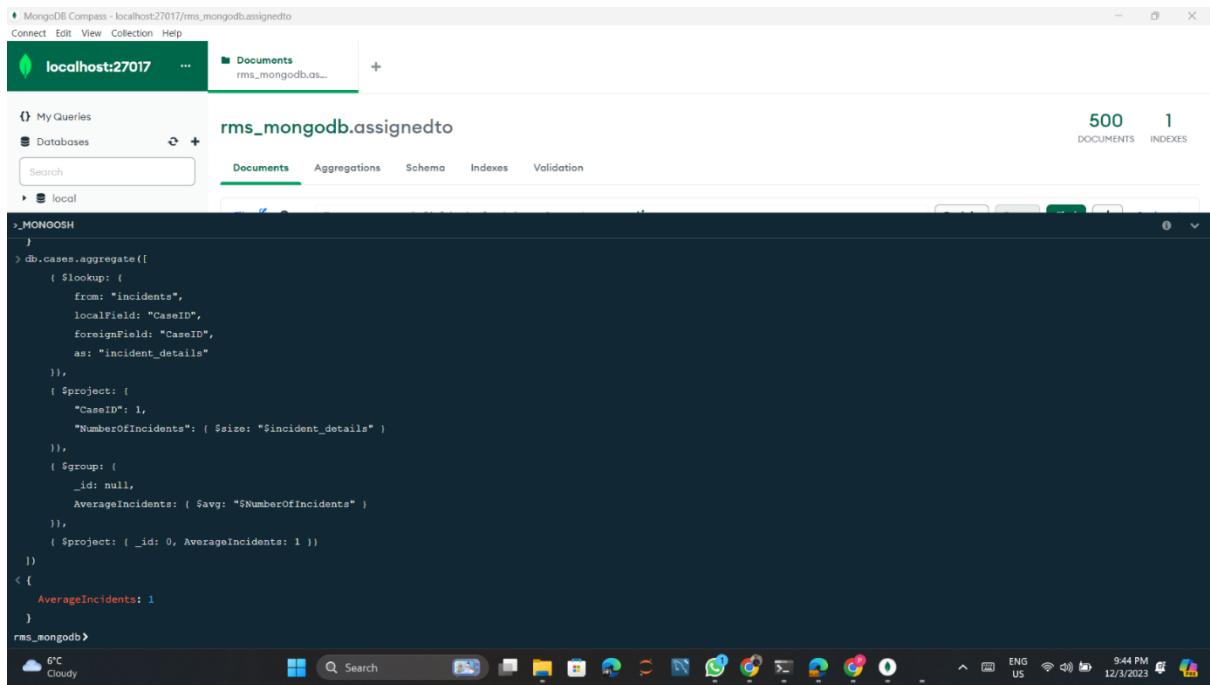
4. Aggregating the types of evidence and count their occurrences in all cases

The screenshot shows the MongoDB Compass interface. The top bar displays "MongoDB Compass - localhost:27017/rms_mongodb.assignedto". The left sidebar shows "My Queries" and "Documents" for the "rms_mongodb.assignedto" collection. The main area shows the aggregation pipeline:

```
> db.evidence.aggregate([
  { $group: {
    _id: "$EvidenceType",
    Count: { $sum: 1 }
  }},
  { $sort: { Count: -1 } }
])
< [
  {
    _id: 'Explosives',
    Count: 52
  },
  {
    _id: 'Financial',
    Count: 52
  },
  {
    _id: 'Document',
    Count: 49
  }
]
```

The status bar at the bottom indicates "Cloudy" weather, "6°C", "ENG US", "9:42 PM", and the date "12/3/2023".

5. Finding the average number of incidents per case



The screenshot shows the MongoDB Compass interface. The top navigation bar includes 'MongoDB Compass - localhost:27017/rms_mongodb.assignedeto', 'Connect', 'Edit', 'View', 'Collection', and 'Help'. Below the navigation is a toolbar with icons for 'Documents' (selected), 'Aggregations', 'Schema', 'Indexes', and 'Validation'. The main area displays the 'rms_mongodb.assignedeto' collection. On the left, there's a sidebar with 'My Queries' and 'Databases' sections, and a search bar. The main pane shows the aggregation pipeline:

```
>_MONOOSH
>
> db.cases.aggregate([
    {
        $lookup: {
            from: "incidents",
            localField: "CaseID",
            foreignField: "CaseID",
            as: "incident_details"
        }
    },
    {
        $project: {
            CaseID: 1,
            "NumberOfIncidents": { $size: "$incident_details" }
        }
    },
    {
        $group: {
            _id: null,
            AverageIncidents: { $avg: "$NumberOfIncidents" }
        }
    },
    {
        $project: { _id: 0, AverageIncidents: 1 }
    }
])
< {
    AverageIncidents: 1
}
rms_mongodb>
```

The status bar at the bottom indicates 'Cloudy' weather, '6°C', 'ENG US', '9:44 PM', '12/3/2023', and a battery icon.

The answer is 1 because we gave unique incidentIDs

6. Application Demonstration

The screenshot shows a Jupyter Notebook running on a Windows desktop. The notebook has a single cell (In [7]) containing Python code to query a MySQL database for suspect and case counts. The output shows a table with two rows: one for 'Yes' and one for 'No'. Below the notebook, the Windows taskbar is visible with various icons and system status.

```
In [7]:  
1 import mysql.connector  
2 import pandas as pd  
3 from sqlalchemy import create_engine  
4  
5 connection = mysql.connector.connect(  
6     host="localhost",  
7     user="root",  
8     password="Iamironman2!",  
9     database="rms1",  
10    auth_plugin='mysql_native_password'  
11 )  
12  
13 if connection.is_connected():  
14     print("Connection established...")  
15 cursor = connection.cursor()  
16  
17 query = """  
18     SELECT CriminalRecord, COUNT(*) as CaseCount  
19     FROM Suspects s  
20     LEFT JOIN Involved i ON s.SuspectID = i.SuspectID  
21     LEFT JOIN Cases c ON i.CaseID = c.CaseID  
22     GROUP BY CriminalRecord  
23 """  
24  
25 engine = create_engine('mysql+mysqldb://root:Iamironman2!@localhost/rms1')  
26 df = pd.read_sql(query, engine)  
27 print(df)  
28 cursor.close()  
29 connection.close()  
  
Connection established..  
CriminalRecord CaseCount  
0 Yes 257  
1 No 243
```

Fig 6.1

The screenshot shows a Jupyter Notebook running on a Windows desktop. The notebook has a single cell (In [8]) containing Python code to connect to a MySQL database and create a bar chart of case statuses. The output shows the command 'Connection established...' and the generated bar chart. Below the notebook, the Windows taskbar is visible with various icons and system status.

```
In [8]:  
1 import mysql.connector  
2 from mysql.connector import Error  
3 import pandas as pd  
4 from sqlalchemy import create_engine  
5 import matplotlib.pyplot as plt  
6  
7  
8 conn = mysql.connector.connect(host='localhost', password='Iamironman2!', user='root')  
9  
10 if conn.is_connected():  
11     print("Connection established...")  
12  
13 connection = mysql.connector.connect(host='localhost',  
14 database='rms1',  
15 user='root',  
16 password='Iamironman2!',  
17 auth_plugin='mysql_native_password')  
18  
19 if connection.is_connected():  
20     print("Connection established...")  
21 cursor = connection.cursor()  
22  
23 cursor.execute("select database();")  
24 record = cursor.fetchone()  
25 print("Your connected to database: ", record)  
26  
27 query = "SELECT CaseStatus, COUNT(*) FROM Cases GROUP BY CaseStatus"  
28  
29 cursor.execute(query)  
30  
31 results = cursor.fetchall()  
32  
33 cursor.close()  
34 connection.close()  
35  
36 case_statuses, counts = zip(*results)  
37  
38 plt.bar(case_statuses, counts, color='blue')  
39 plt.xlabel('Case Status')  
40 plt.ylabel('Number of Cases')  
41 plt.title('Distribution of Case Status')  
42 plt.show()
```

Fig 6.2

```

Connection established...
Connected to MySQL Server version 8.0.35
Your connected to database: ('rms1',)

```

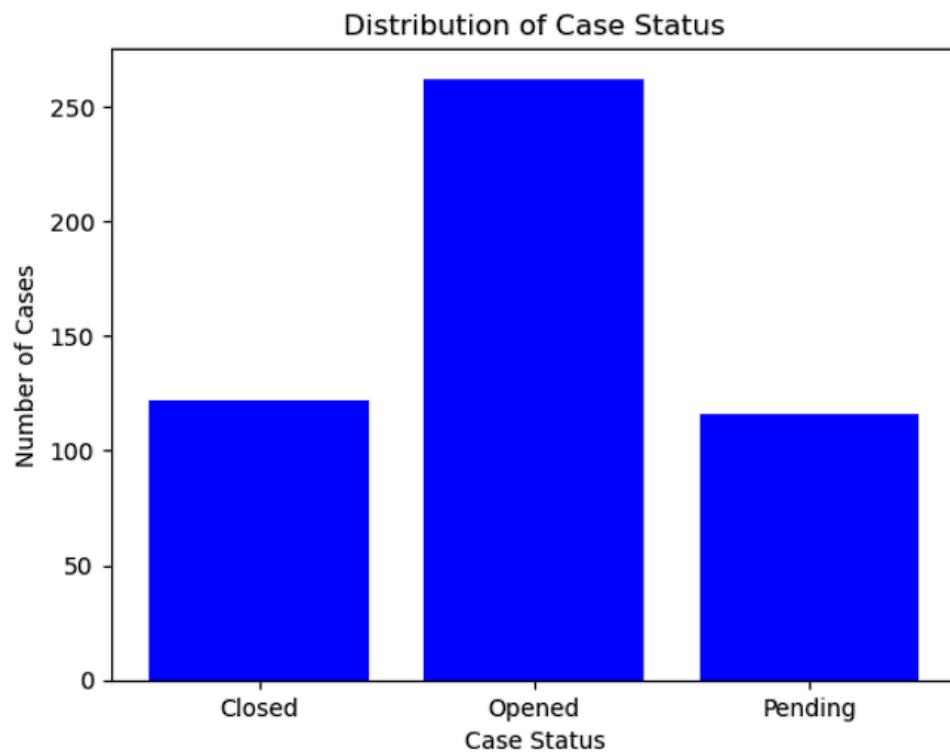
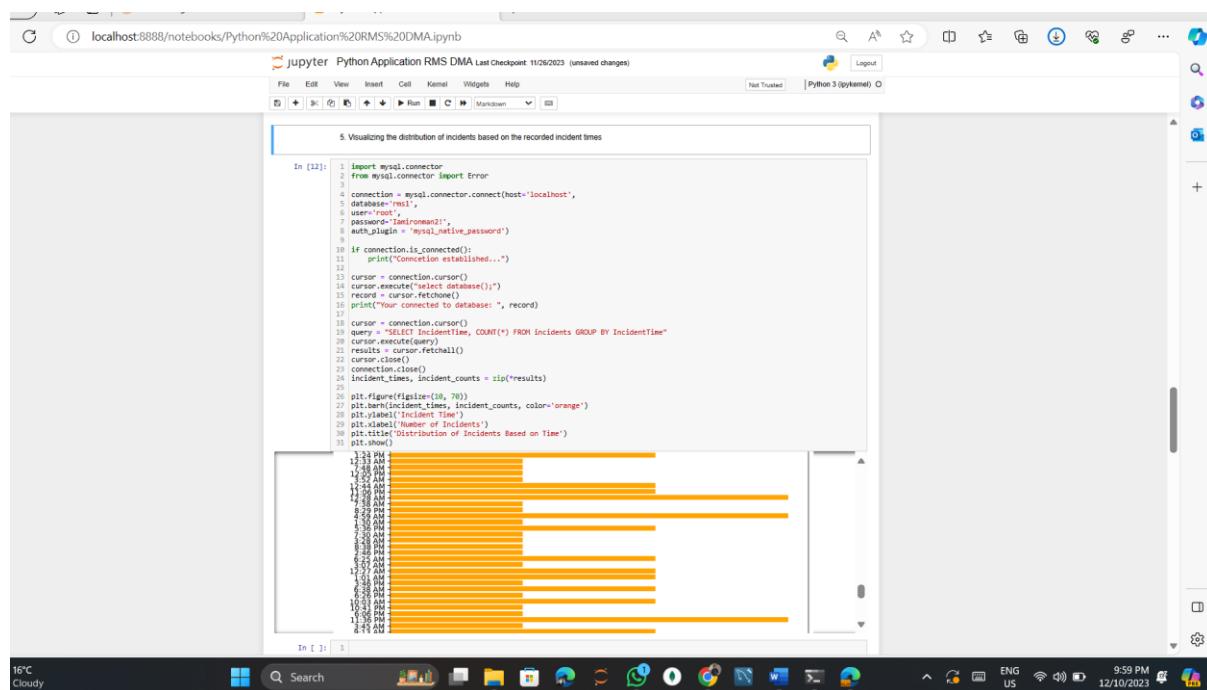


Fig 6.3



```

jupyter Python Application RMS DMA Last Checkpoint: 11/26/2023 (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help
Not Trusted | Python 3 (ipykernel) O
In [5]: 1 import mysql.connector
2 import matplotlib.pyplot as plt
3 import pandas as pd
4
5 connection = mysql.connector.connect(
6     host="localhost",
7     database="rms1",
8     user="root",
9     password="Iamironman21",
10    auth_plugin="mysql_native_password"
11 )
12
13 if connection.is_connected():
14     print("Connection established...")
15
16 query = """
17     SELECT Cases.CaseID, COUNT(DISTINCT AssignedTo.OfficerID) AS AssignedOfficers
18     FROM Cases
19     LEFT JOIN AssignedTo ON Cases.CaseID = AssignedTo.CaseID
20     GROUP BY Cases.CaseID
21 """
22
23 cursor = connection.cursor()
24 cursor.execute(query)
25 results = cursor.fetchall()
26 cursor.close()
27 connection.close()
28
29 df = pd.DataFrame(results, columns=['CaseID', 'AssignedOfficers'])
30
31 plt.figure(figsize=(10, 6))
32 plt.scatter(df['CaseID'], df['AssignedOfficers'], color='red', marker='o')
33 plt.title('Scatter Plot: Number of Assigned Officers per Case')
34 plt.xlabel('Case ID')
35 plt.ylabel('Number of Assigned Officers')
36 plt.show()
37
38 plt.figure(figsize=(10, 6))
39 plt.hist(df['AssignedOfficers'], bins=10, color='green', alpha=0.7)
40 plt.title('Histogram: Distribution of Assigned Officers per Case')
41 plt.xlabel('Number of Assigned Officers')
42 plt.ylabel('Frequency')
43 plt.show()
44
45 officer_counts = df['AssignedOfficers'].value_counts()
46 labels = officer_counts.index
47 sizes = officer_counts.values
48
49 plt.figure(figsize=(10, 6))
50 plt.pie(sizes, labels=labels, autopct='%1.1f%%', colors=['skyblue', 'lightcoral', 'lightgreen'])
51 plt.title('Pie Chart: Distribution of Cases Based on Assigned Officers')
52 plt.show()
53
54 plt.show()

```

Fig 6.5

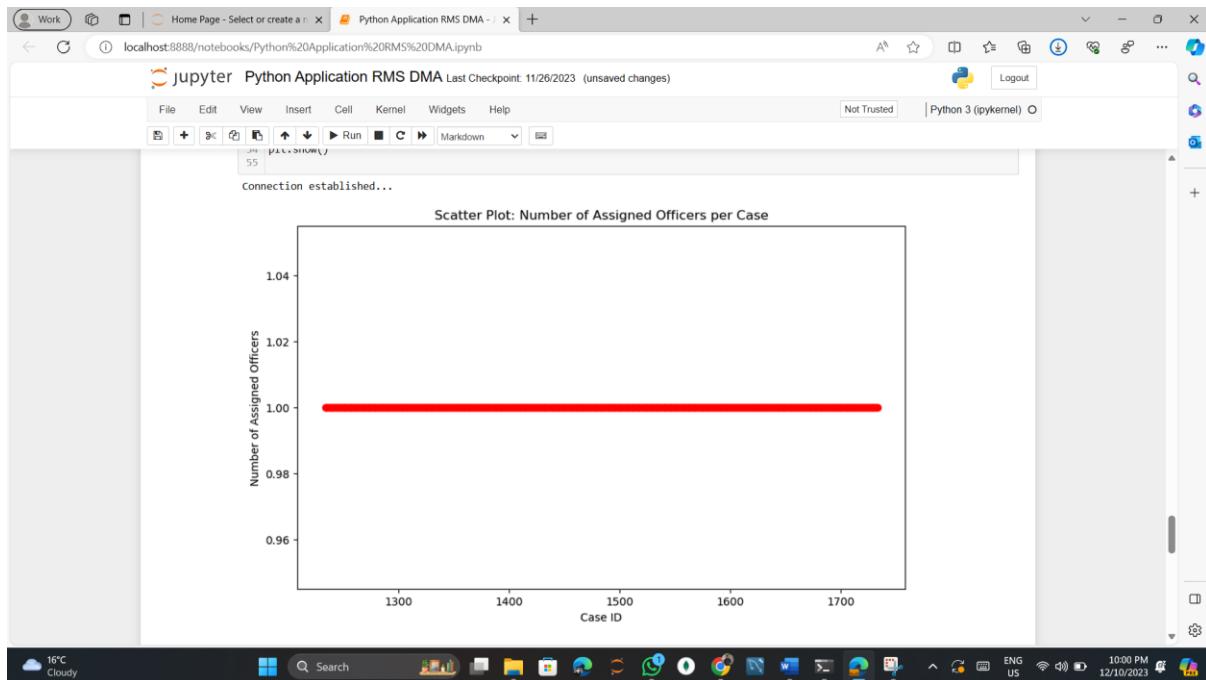


Fig 6.6

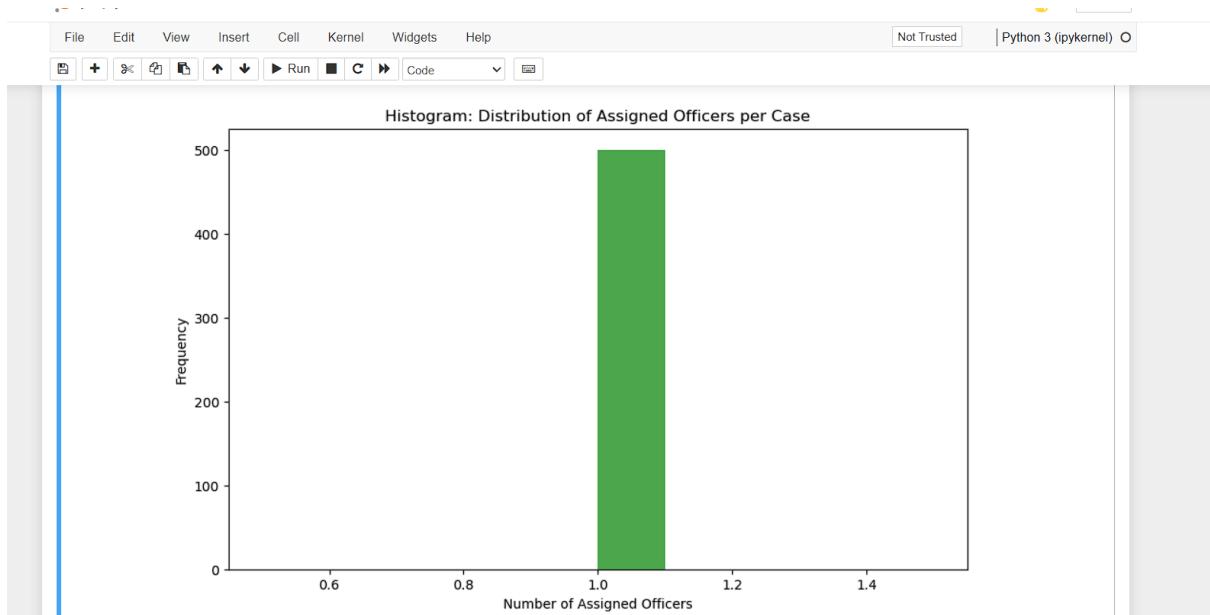


Fig 6.7

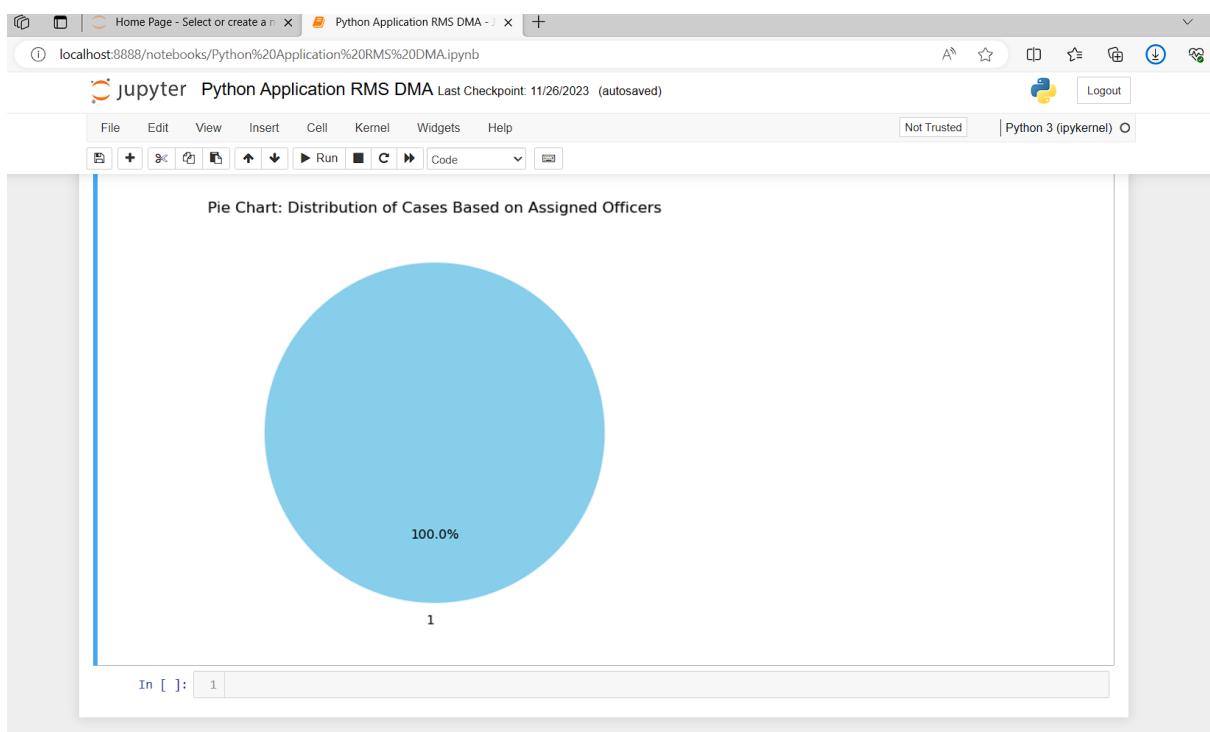


Fig 6.8