Crime Analytics System

Rachana Dharmavaram(rachanad)
Department of Computer Science and Engineering
The State University of NewYork at Buffalo
Buffalo, United States
rachanad@buffalo.edu

Venkata Sri Sai Surya Mandava(mandava6)
Department of Computer Science and Engineering
The State University of NewYork at Buffalo
Buffalo, United States
mandava6@buffalo.edu

***Abstract*— The database-driven Crime Analytics System is designed to effectively retain, analyze, and manage crime-related data. To offer a disciplined method to crime investigation, it combines several elements like crime reports, suspects, police, arrests, locations, cases, evidence, and court proceedings. The system lets law enforcement departments monitor crime trends, spot repeat offenders, and create automatic warnings. While advanced features include triggers for real-time alerts and stored routines for report production improve operational efficiency, it supports complicated searches for crime trend analysis, suspect tracking, and case progress monitoring. Maintaining one-to--one, one-to- many, and many-to- many linkages among entities, the E/R diagram guarantees a well-organized relational model. Through a scalable, safe, and efficient approach to modern crime management, law enforcement agencies can increase public safety, optimize investigations, and enhance decision-making by use of this database.**

Keywords*— *Crime Analytics, Law Enforcement Database, Crime Reports, Suspect Tracking, Arrest Records, Data Integrity, Relational Database, E/R Diagram, Forensic Investigation, Crime Pattern Analysis.

I. Introduction

Title: Crime Analytics System: Improving Law Enforcement with Data-Driven Realizations.

Problem Statement:
Effective management of crime data presents great difficulties for law enforcement departments. Conventional approaches such Excel spreadsheets or paper-based recordkeeping could provide data duplication, discrepancies, and trouble finding pertinent information. Agencies need a system that can efficiently track, evaluate, and forecast crime trends as crime rates vary and events grow more complicated.

Designed to be a centralized crime tracking and analysis database, the Crime Analytics System lets law enforcement keep and access crime reports, follow criminal behavior, and forecast high-risk locations. This system will give law enforcement officials and analysts real-time, ordered access to crime data, therefore enabling informed judgments and enhancement of public safety.

Why do you need a database instead of an Excel file?

Scalability: As crime statistics keep rising, an Excel file becomes unworkable when managing hundreds of records. Large datasets can be effectively maintained in a relational database therefore avoiding performance problems. Structured storage and fast access to crime-related data made possible by Postgres help law enforcement departments to manage an ever-rising caseload over time.

Data Integrity & Consistency: Conventions, relationships, and data validation offered by a database help to stop duplicate or inconsistent records. Excel's absence of built-in enforcement systems makes it easier for users to unintentionally change or erase important data, therefore producing erroneous records. Using normalization, foreign keys, and primary keys guarantees consistency and accuracy in crime records in a database.

Advanced Querying & Analysis: Many times, crime research calls for sophisticated searches include tracking serial offenders, matching suspects to several incidents, and determining crime trends by geography. By supporting SQL searches, joins, and aggregations, a database lets law enforcement departments rapidly access significant insights from unprocessed data. Conversely, Excel makes real-time analysis challenging since it needs hand data manipulation and filtering.

Triggers & Automation: Key chores such alerting when a high-risk person is found, updating criminal records when fresh evidence is entered, and setting off alarms for repeat offenders can all be automated by the criminal Analytics System. Excel lets users manually update records and reports; it does not include event-driven automation.

Access control and security: Sensitive criminal data access is guaranteed by a role-based access control system exclusively for authorized users. Different degrees of access will be granted to police, detectives, and analysts, therefore preventing illegal data leaks or alterations. Conversely, Excel

files run a danger of data abuse since they may be readily changed, duplicated, or shared without security policies.

II. Background and Potential of Project

A. Background of the problem:

Crime is progressively bringing problems for people, communities, and governments. Crime investigation, suspect identification, and prevention of recurrent events fall to law enforcement. Still, accurate tracking of crime figures is challenging work. Many law enforcement departments still depend on Excel spreadsheets, paper records, or antiquated tools unable of sophisticated data processing. With these methods, data is duplicated, unreliable, and ineffectual, hence monitoring criminal activity with them is challenging.

Law enforcement needs a consistent way to keep, access, and examine crime statistics since rates vary and new trends indicate. Without efficient data management, police find repeat offenders, link events, and pinpoint areas with high crime rates difficultly. Investigations take more time; it is more difficult to look at crime tendencies; units find it more difficult to coordinate without an organized database. Without a reliable system, law enforcement agencies cannot effectively deploy their resources, therefore compromising the effectiveness of programs meant to prevent crime and ensure person safety.

B. Significance of the problem:

Two of the biggest challenges in criminal data management are data overload and poor structure. Police departments process hundreds of crimes report annually as well as suspect information and forensic evidence. Without a strong system, these records could be lost, copied, or misplaced, hence influencing delayed investigations and incomplete case files. Methodical organization of crime statistics depends on a centralized database, which also allows police fast access to relevant data.

Still another challenge is not having real-time access to criminal records. High-risk police need to have fast access to evidence reports, past criminal records, and suspect information. Excel-based or hand-made record-keeping does not provide real-time updates, either though. Many times, researchers must depend on outdated information, which can hinder their ability to respond to crimes. Real-time updating of criminal records guaranteed by a database system ensures police accurate and updated information anytime required.

Another main issue is the challenge in crime trend analysis. Data-driven insights let one identify locations prone to crime, project future events, and follow repeat criminals. Creating such reports calls for manual work in conventional systems, which increases error risk and time-consuming nature. Advanced searches, statistical analysis, and automatic report generating made possible by a database assist law enforcement spot trends and act early on.

Lack of interconnectedness among crime records is another restriction of conventional systems. Many crimes connect several suspects, sites, and police. Tracking criminal networks and linking similar cases gets quite challenging without a relational database. Detectives could find it difficult to pinpoint shared suspects between several crimes or examine the connections between several events. A well-organized database helps police to effectively link cases, suspects, and evidence, thereby enhancing the investigative procedures.

At last, data integrity and security are quite important issues. Personal information of victims, witnesses, and suspects included in crime records is quite sensitive. Without appropriate permission, manual records and Excel sheets can be readily changed, erased, or shared, therefore creating security issues and data breaches. Role-based access control (RBAC) made possible by a database system guarantees that entries may be viewed or changed only by authorized users. This guarantees crime record accuracy, enhances data security, and stops illegal alterations.

C. Objectives of the Crime Analytics System:

By use of a safe, scalable, and effective crime management database, the Crime Analytics System seeks to solve these issues. The system primarily aims to:

Centralized Data Storage: The database will guarantee simple access of crime records and act as a single source of truth, therefore removing data duplication.

Real-Time Data Access: Officers and detectives will have fast access to crime reports, suspect information, and case updates—so enhancing reaction times and coordination.

Crime Trend Analysis: The technology will provide sophisticated searches and automatic reports, therefore enabling law enforcement to spot trends, forecast crime-prone locations, and deploy resources most wisely.

Interconnected Crime Records: Linking similar cases, suspects, and evidence will enable investigators to trace criminal activity and identify organized crime networks via interconnected crime records.

Enhanced Security and Access Control: The system will provide role-based access, therefore stopping illegal changes and safeguarding of private crime information.

D. Potential of the project:

By use of a consolidated and structured method of crime data handling, the Crime Analytics System improves law enforcement. Conventional approaches such spreadsheets and paper-based records are ineffective, which makes trend analysis challenging and resource allocation problematic as well as tracking repeat offenders challenging. Real-time access to crime statistics made possible by this system will help investigations, reaction times, public safety to be improved.

Its capacity to examine crime trends and forecast high-risk locations makes a significant impact as well Processing past crime statistics helps authorities to pinpoint crime hotspots, peak hours, and trends thereby enabling effective resource allocation and preventative action. Predictive analytics enable crime prevention before they start, not reaction following events.
The approach also improves tracking of returning criminals and detection of criminal networks. Many crimes involve historical histories, hence physically obtaining this evidence takes time. Faster inquiries, links between crimes, and quick access to suspect profiles made possible by the database help to enable.

Through local, state, and federal agencies exchanging crime data for quick case conclusion, the system also enhances inter-agency cooperation. Automated alarms notify law enforcement of repeat offenders or offenses in high-risk regions, therefore improving response times and initiatives at crime prevention.

Apart from law enforcement, the system supports data-driven policymaking by evaluating law enforcement effectiveness, crime trends, and demands for resource allocation.

These insights enable legislators to implement projects aiming at crime prevention and strengthen public safety strategies. Security is still another great advantage. Unlike standard spreadsheets, which are prone to unauthorized alterations, the Crime Analytics System incorporates role-based access control (RBAC) to defend sensitive information and retain accuracy.

The Crime Analytics System will eventually force law enforcement into more proactive, data-driven, effective behavior. By means of trend analysis, tracking of offenders, enhanced cooperation, real-time alerts, and policy support, the system will increase public safety and crime prevention.

III. Target User

A. Target User:

Designed for law enforcement agencies, government authorities, and crime analysts looking for a methodical approach to retain, access, and examine crime data, the Crime Analytics System is by means of real-time access to crime records, offender profiles, and investigative data, so enabling law enforcement to make informed decisions and improve efforts at crime prevention.

B. Primary Users:

Law Enforcement Authorities: The system will let police officials and detectives file crime reports, monitor investigations, and find repeat offenders. Real-time crime data lets them rapidly access case details, suspect information, and crime locations, therefore enabling faster and more effective responses to events.

Crime Investigators & Analysts: Analysts working with crime investigators and analysts will use the technology to find criminal trends, project high-risk locations, and spot patterns. By creating crime heat maps, examining recurring offenders, and tracking suspect movement, they can let law enforcement departments maximize their resources.

Government Officials & Policymakers: Policymakers and government officials from cities can examine crime data using the technology and distribute funds accordingly. If a neighborhood's crime rate is rising, for instance, they can boost police patrols, fund community projects, or bring in fresh crime prevention techniques.

C. Database Administrators:

A selected IT team or law enforcement data analysts will be in charge of the database. They are in charge of controlling who can access what, keeping data safe, making regular backups, and making sure the system runs at its best. Role-based access control (RBAC) will be used by administrators to make sure that only people who are allowed to can see or change private data.

D. Real-Life Scenario:

The Crime Analytics System is used by a metropolitan police force to make reporting and analyzing crimes easier. When a police officer on duty reacts to a robbery, they enter the information about the case into the system. The system checks instantly for past crimes that happened in the same place and lets police know if a suspect looks like a known repeat offender.

At the same time, crime experts look at patterns in the area and see that there are more thefts at night in certain neighborhoods. Because of these findings, the department sends more police to areas with a lot of crime. At the same time, city officials use crime data to pay for better streetlights and programs that make communities safer, which helps lower crime over time.

IV. E/R Diagram & Relations

Fig. 1. Entity Relationship Diagram for Crime Analytics System Database.

A. Realtionships between different tables:

Arrests → Suspects (Many-to-One)
Every arrest relates to one suspect; however, a suspect could be arrested many times.

Arrests → CrimeReports (Many-to-One)
Though many arrests might be connected to the same crime, every arrest is connected to one particular crime report.

Arrests → Officers (Many-to-One)
One officer makes each arrest; yet one officer may make many arrests.

CrimeReports → Officers (Many-to-One)
Each officer files a crime report, but an officer might file many crime reports.

CrimeReports → Locations (Many-to-One)
While each crime report may involve multiple locations, each crime report is done at one location.

CrimeReports → Cases (One-to-Many)
While a crime report may relate to multiple legal cases, each legal case is based on one crime report.

CrimeReports → Witnesses (One-to-Many)
While each witness is related to one crime report, one crime may have many witnesses.

Cases → Suspects (Many-to-One)
Each legal case relates to one suspect, but one suspect may be related to multiple legal cases.

Cases → Evidence (One-to-Many)
While each piece of evidence relates to one legal case, one legal case may have many pieces of evidence.

Cases → CourtCases (One-to-One)
Each legal case generates one court case (court file), and each court case relates to specifically one legal case.

Cases → Victims (One-to-Many)
Each victim is related to one specific legal case, but one legal case may have many victims.

Victims → CrimeReports (Many-to-One)
While a single crime report may have numerous victims, each victim is related to one crime report.

OfficerBadge → Officers (Many-to-One)
Each badge record relates to one officer, but an officer may have numerous badge records, that is historical records.

B. List of Relations and their Attributes.

a) CrimeReports:

Purpose: Store information on recorded offenses.
Primary Key: CrimeID is the Primary key since every crime report has a different identification.
Foreign Key:
LocationID → References Locations(LocationID) (guarantees every crime is connected to a legitimate location).
OfficerID → References Officers(OfficerID) (assigns an officer to the case).

TABLE I. TABLE CRIMEREPORTS

Attribute	Data Type	Constraints	Purpose	Default
CrimeID	SERIAL	Primary Key, NOT NULL	Each crime report has its own unique identifier.	Auto-increment
Date	DATE	NOT NULL	Crime reported date	None
Type	VARCHAR(50)	NOT NULL	Crime type e.g.: assault, robbery	None
LocationID	INT	Foreign Key REFERENCES Locations(LocationID) ON DELETE CASCADE	Links the locations where crime happened	None
OfficerID	INT	Foreign Key REFERENCES Officers(OfficerID) ON DELETE SET NULL	Assigned officer for crime investigation	NULL if officer is removed

b) Suspects:

Purpose: Store data on suspects.
Primary Key: SuspectID (every suspect has a different identifier).
Foreign Key: No Foreign Key

TABLE II. TABLE SUSPECTS

Attribute	Data Type	Constraints	Purpose	Default
SuspectID	SERIAL	Primary Key, NOT NULL	Each suspect has their own unique identifier.	Auto-increment
Name	VARCHAR(100)	NOT NULL	Suspect's Full Name	None
Age	INT	CHECK (Age > 0)	Suspect's Age	None
Criminal History	TEXT	NULLABLE	History of previous crimes	NULL

c) Officers:

Purpose: Preserves information regarding law enforcement personnel.
Primary Key: OfficerID (Each officer possesses a distinct identifier).

Foreign Keys:
DepartmentID → References Departments (DepartmentID)
(Ensures that officers are affiliated with legitimate departments).

TABLE III. TABLE OFFICERS

Attribute	Data Type	Constraints	Purpose	Default
OfficerID	SERIAL	Primary Key, NOT NULL	Each Officer has their own unique identifier.	Auto-increment
Name	VARCHAR(100)	NOT NULL	Officer's Full Name	None
BadgeNumber	VARCHAR(20)	UNIQUE, NOT NULL	Unique badge number of officer	None
DepartmentID	INT	Foreign Key REFERENCES Departments (DepartmentID) ON DELETE SET NULL	Officer's Department	NULL if department is removed

d) Locations:

Purpose: Stores information about criminal sites.
Primary Key: LocationID (every site has a different identification).
Foreign Key: No Foreign Key

TABLE IV. TABLE LOCATIONS

Attribute	Data Type	Constraints	Purpose	Default
LocationID	SERIAL	Primary Key, NOT NULL	Each Location has its own unique identifier.	Auto-increment
Address	TEXT	NOT NULL	Full address where the crime happened	None
City	VARCHAR(50)	NOT NULL	Name of the city	None
ZipCode	VARCHAR(10)	NOT NULL	Zip Code of the location	None

e) Cases:

Purpose: Links crimes to suspects and follows case status.
Primary Key: Case ID (Every case has a distinct ID) .
Foreign keys:
CrimeID → References CrimeReports(CrimeID) (ensures every case links to a crime).
Suspect ID → References Suspects (Ensues case has a suspect).

TABLE V. TABLE CASES

Attribute	Data Type	Constraints	Purpose	Default
CaseID	SERIAL	Primary Key, NOT NULL	Each Case has its own unique identifier.	Auto-increment
CrimeID	INT	Foreign Key REFERENCES CrimeReports(CrimeID) ON DELETE CASCADE	Links the case to a crime	None

f) Victims:

Purpose: Store victim data for use.
Primary Key: Victim ID (Every victim has a distinct ID).
Foreign Keys:
CrimeID → References CrimeReports(CrimeID) (Links victim to crime)

TABLE VI. TABLE VICTIMS

Attribute	Data Type	Constraints	Purpose	Default
VictimID	SERIAL	Primary Key, NOT NULL	Each Victim has their own unique identifier.	Auto-increment
Name	VARCHAR(100)	NOT NULL	Full name of the victim	None
Age	INT	CHECK (Age > 0)	Victim's age	None
CrimeID	INT	Foreign Key REFERENCES CrimeReports(CrimeID) ON DELETE CASCADE	Links victim to crime	None

g) Evidence:

Purpose: Store case-related documentation for evidence.
Primary Key: EvidenceID (every evidence item has a distinct identification).
Foreign Keys:
CaseID → References Cases(CaseID) (ensures evidence links a case).

TABLE VII. TABLE EVIDENCE

Attribute	Data Type	Constraints	Purpose	Default
EvidenceID	SERIAL	Primary Key, NOT NULL	Each evidence has its own unique identifier.	Auto-increment

CaseID	INT	Foreign Key REFERENCES Cases(CaseID) ON DELETE CASCADE	Links evidence to a case	None
Description	TEXT	NOT NULL	Detailed description about the evidence	None
Collected Date	DATE	NOT NULL	Date of evidence collection	None

h) Witnesses:

Purpose: Store witnesses' information for use.
Primary Key: WitnessID(every witness has a unique identification).
Foreign keys:
Crime ID → References CrimeReports(CrimeID) (Links witness to crime).

TABLE VIII. TABLE WITNESSES

Attribute	Data Type	Constraints	Purpose	Default
WitnessID	SERIAL	Primary Key, NOT NULL	Each witness has their own unique identifier.	Auto-increment
Name	VARCHAR(100)	NOT NULL	Full name of the witness	None
Statement	TEXT	NOT NULL	Detailed statement given witness	None
CrimeID	INT	Foreign Key REFERENCES CrimeReports(CrimeID) ON DELETE CASCADE	Links witness to crime	None

i) Arrests:

Purpose: Stores specifics about suspected arrests.
Primary Key: ArrestID (every arrest has a different identification).
Foreign Keys:
Suspect ID → References Suspects(SuspectID) (Links arrest to a suspect).
CrimeID → References CrimeReports(CrimeID) (guarantees that an arrest corresponds with a crime).
Officer ID → References Officers(OfficerID) (Notes the officer in charge of the arrest).

TABLE IX. TABLE ARRESTS

Attribute	Data Type	Constraints	Purpose	Default
ArrestID	SERIAL	Primary Key, NOT NULL	Each arrest has its own unique identifier.	Auto-increment
SuspectID	INT	Foreign Key REFERENCES Suspects(SuspectID) ON DELETE CASCADE	Links arrest to a suspect	None
CrimeID	INT	Foreign Key REFERENCES CrimeReports(CrimeID) ON DELETE CASCADE	Links arrest to a crime	None
ArrestDate	DATE	NOT NULL	Arrest Date	None

j) CourtCases:

Purpose: Track legal cases involving crimes.
Primary Key: CourtCaseID (every court case has a distinct ID).
Foreign keys:
CaseID → References Cases(CaseID) (Links the court case to a criminal case).

TABLE X. TABLE COURTCASES

Attribute	Data Type	Constraints	Purpose	Default
CourtCaseID	SERIAL	Primary Key, NOT NULL	Each court case has its own unique identifier.	Auto-increment
CaseID	INT	Foreign Key REFERENCES Cases(CaseID) ON DELETE CASCADE	Links arrest to a suspect	None
JudgeName	VARCHAR(100)	NOT NULL	The name of the judge who is hearing the case	None
Verdict	VARCHAR(50)	CHECK (Verdict IN ('Guilty', 'Not Guilty', 'Pending'))	Final decision of the court	None

V. TRANSFORMATIONS

A. Functional Dependencies:

a) CrimeReports:

CrimeReports (CrimeID, Date, Type, LocationID, OfficerID)

Functional Dependencies:
CrimeID → Date, Type, LocationID, OfficerID
Here CrimeID finds all other properties uniquely.
This CrimeReports relation is already in BCNF
as CrimeID is a superkey for this relation.

b) Suspects:

Suspects (SuspectID, Name, Age, CriminalHistory)

Functional Dependencies:
SuspectID → Name, Age, CriminalHistory
Here SuspectID finds all other properties uniquely.
This Suspects relation is already in BCNF
as SuspectID is a superkey for this relation.

c) Officers:

Officers(OfficerID,Name,BadgeNumber, DepartmentID)

Functional Dependencies:
OfficerID → Name, BadgeNumber, DepartmentID
BadgeNumber → OfficerID, Name, DepartmentID

Here BadgeNumber finds all other properties uniquely.
This Officers relation is not in BCNF as BadgeNumber is not a superkey for this relation, but it is a unique identifier.
Decomposition is required for this Officers relation. We can split this relation into two relations named Officers and OfficerBadge.

Required Decompositions are:
Officers (OfficerID, Name, DepartmentID)
OfficerBadge (BadgeNumber, OfficerID)

d) Locations:

Locations (LocationID, Address, City, ZipCode)

Functional Dependencies:
LocationID → Address, City, ZipCode
This Locations relation is already in BCNF
as LocationID is a superkey for this relation.

e) Cases:

Cases (CaseID, CrimeID, SuspectID, Status)

Functional Dependencies:
CaseID → CrimeID, SuspectID, Status
This Cases relation is already in BCNF as CaseID is a superkey for this relation.

f) Victims:

Victims (VictimID, Name, Age, CrimeID)

Functional Dependencies:
VictimID → Name, Age, CrimeID
This Victims relation is already in BCNF as VictimID is a superkey for this relation.

g) Evidence:

Evidence(EvidenceID,CaseID,Description, CollectedDate)

Functional Dependencies:
EvidenceID → CaseID, Description, CollectedDate
This Evidence relation is already in BCNF
as EvidenceID is a superkey for this relation.

h) Witnesses:

Witnesses (WitnessID, Name, Statement, CrimeID)

Functional Dependencies:
WitnessID → Name, Statement, CrimeID
This Witnesses relation is already in BCNF
as WitnessID is a superkey for this relation.

i) Arrests:

Arrests (ArrestID, SuspectID, CrimeID, ArrestDate, OfficerID)

Functional Dependencies:
ArrestID →SuspectID,CrimeID,ArrestDate, OfficerID
This Arrests relation is already in BCNF as ArrestID is a superkey for this relation.

j) CourtCases:

CourtCases(CourtCaseID,CaseID,JudgeName, Verdict)

Functional Dependencies:
CourtCaseID → CaseID, JudgeName, Verdict
This CourtCases relation is already in BCNF
as CourtCaseID is a superkey for this relation.

VI. IMPLEMENTATION OF SQL QUERIES AND TESTING

A. Insertion, Updation, Deletion Queries:

a) Inserting new location:

Query:

```
INSERT INTO Locations (LocationID, Address, City,
ZipCode)
VALUES (3001, '987 Neo Ave', 'Cyberville', '99999');
```

Description of this query: Adds a location with ID 3001 at '987 Neo Ave' , in 'Cyberville' , with '99999' as a postal code.

Implementation:

```
INSERT INTO Locations (LocationID, Address, City, ZipCode)
VALUES (3001, '987 Neo Ave', 'Cyberville', '99999');
```

Data Output | Messages | Notifications

```
INSERT 0 1

Query returned successfully in 49 msec.
```

Fig. 2. Inserting new location.

b) Inserting a new Officer:

Query:

INSERT INTO Officers (OfficerID, Name, DepartmentID)
VALUES (3001, 'Officer Quantum', 9);

Description of this query: Inserts an officer, Officer Quantum, under department ID 9 and unique officer ID 3001.

Implementation:

```
INSERT INTO Officers (OfficerID, Name, DepartmentID)
VALUES (3001, 'Officer Quantum', 9);
```

Data Output | Messages | Notifications

```
INSERT 0 1

Query returned successfully in 55 msec.
```

Fig. 3. Inserting a new Officer.

b) Updating Suspects :

Query:

UPDATE Suspects
SET Age = 50
WHERE SuspectID = 5;

Description of this query: Sets the age of the suspect with an Id of 5 to 50 years, overriding any previous value.

Implementation:

```
UPDATE Suspects
SET Age = 50
WHERE SuspectID = 5;
```

Data Output | Messages | Notifications

```
UPDATE 1

Query returned successfully in 57 msec.
```

Fig. 4. Updation of Suspects.

c) Updating Cases:

Query:

UPDATE Cases
SET Status = 'Pending'
WHERE CaseID = 20;

Description of this query: Changes the case status of caseID = 20 to 'Pending', which typically means the case is suspended or waiting for additional action to be taken.

Implementation:

```
UPDATE Cases
SET Status = 'Pending'
WHERE CaseID = 20;
```

Data Output | Messages | Notifications

```
UPDATE 1

Query returned successfully in 52 msec.
```

Fig. 5. Updation of Cases.

d) Deletion from Victims:

Query:

DELETE FROM Victims
WHERE VictimID = 10;

Description of this query: Permanently removes from the database the victim record for VictimID = 10.

Implementation:

```
DELETE FROM Victims
WHERE VictimID = 10;
```

Data Output | Messages | Notifications

```
DELETE 1

Query returned successfully in 128 msec.
```

Fig. 6. Deletion from Victims.

d) Deletion from Evidence:

Query:

DELETE FROM Evidence
WHERE EvidenceID = 11;

Description of this query: Deletes the evidence item that has an ID of 11 from the system.

Implementation:

```
DELETE FROM Evidence
WHERE EvidenceID = 11;
```

Data Output | Messages | Notifications

```
DELETE 1

Query returned successfully in 45 msec.
```

Fig. 7. Deletion from Evidence.

B. Selection Queries:

a)Using JOIN:

Query:

SELECT cr.CrimeID, cr.Date, cr.Type, o.Name AS OfficerName, l.City

FROM CrimeReports cr
JOIN Officers o ON cr.OfficerID = o.OfficerID
JOIN Locations l ON cr.LocationID = l.LocationID
LIMIT 10;

Description of this query: This query shows how to obtain crime-related data by joining the Crime Reports, Officers, and Locations tables. The query is only limited to 10 records, and it pulls out the crime id, date, kind of crime, officer's name, and city of crime.

Implementation:

```
SELECT cr.CrimeID, cr.Date, cr.Type, o.Name AS OfficerName, l.City
FROM CrimeReports cr
JOIN Officers o ON cr.OfficerID = o.OfficerID
JOIN Locations l ON cr.LocationID = l.LocationID
LIMIT 10;
```

Data Output | Messages | Notifications — Showing rows: 1 to 10 — Page No:

	crimeid integer	date date	type character varying (50)	officername character varying (100)	city character varying (50)
1	1	2025-01-21	Assault	Janice Martin	East Courtneybury
2	2	2025-02-19	Robbery	Adam Roberts	Melissamouth
3	3	2024-10-13	Robbery	Cameron Medina	North Sarah
4	4	2024-04-22	Fraud	Brittany Donaldson	North Christopher
5	5	2023-09-03	Assault	Jon Lee	Danielmouth
6	6	2024-07-30	Theft	Jonathan Flowers	Davidberg
7	7	2024-11-08	Robbery	Brooke Campbell	Lake Darrellberg
8	8	2024-12-05	Theft	Scott Davis	North Nicole
9	9	2024-01-23	Fraud	Monique Phillips	Port Chris
10	10	2024-02-07	Fraud	Maria Martin	Lake Jamestown

Fig. 8. Selection query using JOIN.

b)Using ORDERBY:

Query:
SELECT Name, Age FROM Suspects
ORDER BY Age DESC
LIMIT 10;

Description of this query: This is a query that returns the values of the suspects in the Suspects table which includes the names and ages of suspects, ordered by age in decreasing order. Displaying the 10 oldest people identifies concerning age distribution anomalies.

Implementation:

```
SELECT Name, Age FROM Suspects
ORDER BY Age DESC
LIMIT 10;
```

Data Output | Messages | Notifications

	name character varying (100)	age integer
1	Gregory Miller	70
2	Pamela Young	70
3	Kenneth Weber	70
4	Michael Davis	70
5	James Villa	70
6	Russell Howard	70
7	William Erickson	70
8	Jessica Dyer	70
9	Wesley Carter	70
10	Seth Choi	70

Fig. 9. Selection query using ORDERBY.

c)Using GROUPBY:

Query:
SELECT Type, COUNT(*) AS CrimeCount
FROM CrimeReports
GROUP BY Type;

Description of this query: With the use of the COUNT() method, this query counts in total how many crimes of each type there are, and then groups them by Type (e.g. Fraud, Assault). It is helpful to figure out how frequently different types of crimes occur from the collection.

Implementation:

```
SELECT Type, COUNT(*) AS CrimeCount
FROM CrimeReports
GROUP BY Type;
```

Data Output | Messages | Notifications

	type character varying (50)	crimecount bigint
1	Fraud	724
2	Assault	741
3	Theft	782
4	Robbery	753

Fig. 10. Selection query using GROUPBY.

d)Using Subquery:

Query:
SELECT Type, COUNT(*) AS CrimeCount
FROM CrimeReports
GROUP BY Type;

Description of this query: With the use of the COUNT() method, this query counts in total how many crimes of each type there are, and then groups them by Type (e.g. Fraud, Assault). It is helpful to figure out how frequently different types of crimes occur from the collection.

Implementation:

```
SELECT Name FROM Suspects
WHERE SuspectID IN (
    SELECT SuspectID FROM Cases
    GROUP BY SuspectID
    HAVING COUNT(*) > 2
)
LIMIT 10;
```

Query | Query History

Data Output | Messages | Notifications

	name character varying (100)
1	Jason Fisher
2	Laurie Le
3	Andrew Perry
4	Jesse Green
5	Jessica Odonnell
6	David Richardson
7	Andrew Ayala
8	Alice Miller
9	Nathaniel Hughes
10	Mary Michael

Fig. 11. Selection query using SUBQUERY.

C.Stored Procedures for Basic Operations:

a) Insert Operation:

Query:

```
CREATE OR REPLACE FUNCTION
insert_officer_func(officer_id INT, full_name VARCHAR,
dept_id INT)
RETURNS TEXT AS $$
BEGIN
   INSERT INTO Officers (OfficerID, Name,
DepartmentID)
   VALUES (officer_id, full_name, dept_id);

   RETURN 'Officer inserted successfully';
EXCEPTION WHEN unique_violation THEN
   RETURN 'Officer ID already exists';
END;
$$ LANGUAGE plpgsql;

SELECT insert_officer_func(3001, 'John Doe', 99);
```

Description of this query: The Officers table receives a new officer when this saved function is called. It has exception handlings that will send a custom message if a duplicate OfficerID is found. This prevents data entry errors from happening.

Implementation:

Fig. 12. Stored Procedures for Insert Operation.

b) Update Operation:

Query:

```
CREATE OR REPLACE FUNCTION
update_case_status_func(case_id INT, new_status
VARCHAR)
RETURNS TEXT AS $$
BEGIN
   UPDATE Cases
   SET Status = new_status
   WHERE CaseID = case_id;

   IF FOUND THEN
      RETURN 'Case status updated successfully';
   ELSE
      RETURN 'Case ID not found';
   END IF;
END;
$$ LANGUAGE plpgsql;

SELECT update_case_status_func(599, 'Pending');
```

Description of this query: A status change is made to a case in the Cases table by this process. If it finds and changes a case, it will return a success message. If it cannot find a case, it will send a message saying that CaseID does not exist.

Implementation:

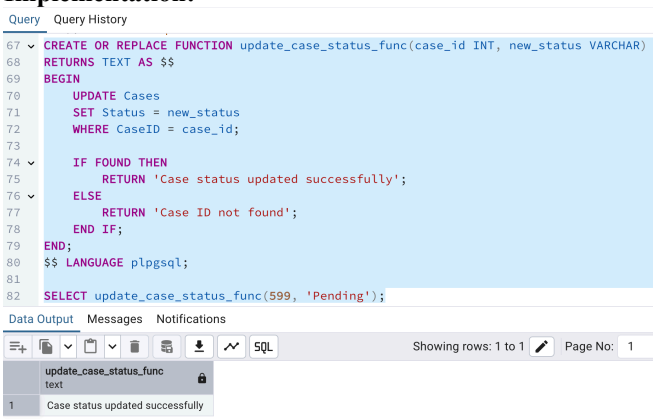

Fig. 13. Stored Procedures for Update Operation.

c) Delete Operation:

Query:

```
CREATE OR REPLACE FUNCTION delete_case_func(cid
INT)
RETURNS TEXT AS $$
BEGIN
   DELETE FROM Cases
   WHERE CaseID = cid;

   IF FOUND THEN
      RETURN 'Case deleted successfully';
   ELSE
      RETURN 'Case ID not found';
   END IF;
END;
$$ LANGUAGE plpgsql;

SELECT delete_case_func(599);
```

Description of this query: After giving a CaseID, this method will remove a case from the Cases table. It sends back either a confirmation message that the delete worked, or a message telling the user that the case could not be found.

Implementation:

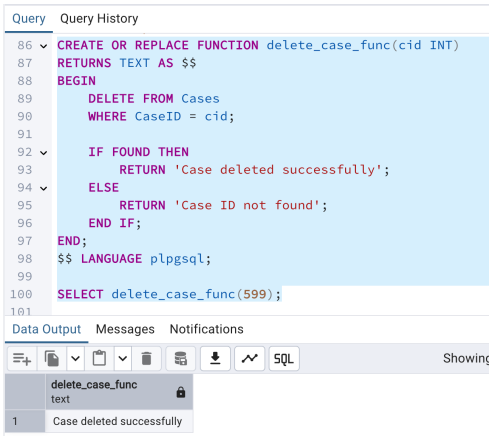

Fig. 14. Stored Procedures for Delete Operation.

d) Select Operation:

Query:

```
CREATE OR REPLACE FUNCTION
get_crimes_by_city(city_name VARCHAR)
RETURNS TABLE(CrimeID INT, Type VARCHAR, Date
DATE, City VARCHAR) AS $$
BEGIN
   RETURN QUERY
   SELECT cr.CrimeID, cr.Type, cr.Date, l.City
   FROM CrimeReports cr
   JOIN Locations l ON cr.LocationID = l.LocationID
   WHERE l.City = city_name;
END;
$$ LANGUAGE plpgsql;

SELECT * FROM get_crimes_by_city('East Dawn');
```

Description of this query: This query returns a list of crimes that have been filtered with a city name. It joins the CrimeReports and Locations tables to provide the relevant fields: CrimeID, Type, Date, and City, which allow for local-specific crime reporting.

Implementation:

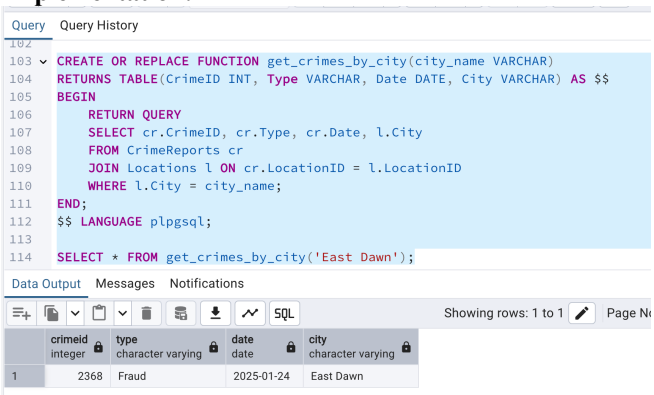

Fig. 15. Stored Procedures for Select Operation.

VII. SIMPLE TRANSCATION HANDLING USING TRIGGER

a) Setting up a failure log table to keep track on transactions:

Query:

```
CREATE TABLE IF NOT EXISTS TransactionFailures (
   LogID SERIAL PRIMARY KEY,
   Message TEXT NOT NULL,
   Timestamp TIMESTAMP DEFAULT NOW()
);
```

Description of query: This query creates a table called TransactionFailures, which keeps a record of errors that happen when an transaction fails. LogID is the auto-incremented primary key, Message will contain the failure reason, and Timestamp is where we record the failure time using NOW(). The IF NOT EXISTS clause checks that the table is only created if it doesn't already exist. This table can be useful in isolating and repairing transactional issues with failed transactions by being able to track and analyze what went wrong.

Implementation:

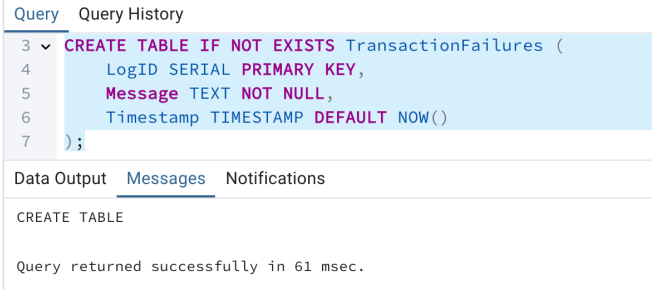

Fig. 16. Creates a table called TransactionFailures.

b) Developing a Trigger Function:

Query:

```
CREATE OR REPLACE FUNCTION log_failed_badge()
RETURNS TRIGGER AS $$
DECLARE
   exists_count INT;
BEGIN
   SELECT COUNT(*) INTO exists_count
   FROM OfficerBadge
   WHERE BadgeNumber = NEW.BadgeNumber;

   IF exists_count > 0 THEN
      INSERT INTO TransactionFailures (Message)
      VALUES ('Duplicate BadgeNumber attempted: ' ||
NEW.BadgeNumber);

      RAISE EXCEPTION 'Duplicate BadgeNumber
Detected: %', NEW.BadgeNumber;
   END IF;

   RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Description of query:

The function log_failed_badge() is implemented to detect and respond to duplicate badge entries in the OfficerBadge table. It counts entries for the new BadgeNumber; if there is at least one, it logs a message to the TransactionFailures table and raises an exception that causes the transaction to abort. That way, you know there will be no new duplicate badge numbers inserted, and you also have evidence of the failure for monitoring and debugging later.

Implementation:

Fig. 17. Creates a Trigger Function.

c) Writing a Trigger for a Pre-Insertion Validation:

Query:

```
DROP TRIGGER IF EXISTS badge_insert_trigger ON OfficerBadge;

CREATE TRIGGER badge_insert_trigger
BEFORE INSERT ON OfficerBadge
FOR EACH ROW
EXECUTE FUNCTION log_failed_badge();
```

Description of query: This SQL query defines a trigger badge_insert_trigger on the OfficerBadge table. The trigger has the timing event specified as PRE, meaning that it will fire (execute) before every insert statement, so that it can validate every new row. A trigger does not execute manually, it instead executes the function log_failed_badge() that we just defined. This function looks for duplicate badge IDs and if any exist, it will log that the operation failed to an audit table then raise an exception preventing the invalid data from being inserted. The DROP TRIGGER IF EXISTS statement will cleanly replace the trigger if it already exists and maintain a full validation approach.

Implementation:

Fig. 18. Creating a Trigger for Pre-Insertion Validation.

d)Implementing a successful transaction:

Query:

```
INSERT INTO Officers (OfficerID, Name, DepartmentID)
VALUES (3100, 'Officer TestSubject', 88);
INSERT INTO OfficerBadge (BadgeNumber, OfficerID)
VALUES ('BN99999', 3100);
```

Description of query: Executing a Successful Transaction
A transaction begins where:
A new officer (ID: 3100) is inserted.
A badge number BN99999 is assigned to this officer.

Implementation:

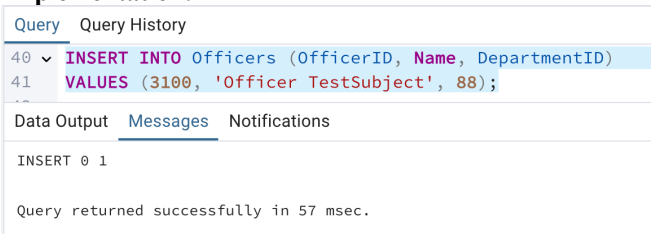

Fig. 19. Insertion a new Officer with ID 3100.

Fig. 20. Insertion a OfficerBadge (BadgeNumber) with value BN99999.

Both insertions are successful since the badge number is unique.

e) Triggering a Failure with a Duplicate Badge:

Query:

```
INSERT INTO Officers (OfficerID, Name, DepartmentID)
VALUES (4002, 'Officer Duplicate', 88);
INSERT INTO OfficerBadge (BadgeNumber, OfficerID)
VALUES ('BN99999', 4002);
```

Description of query: Another transaction starts with the intent to:
Insert a second officer (OfficerID = 4002).
Assign the same badge BN99999 again.
The first insert will run successfully. However, the second insert will fire the trigger. The trigger will record the error in TransactionFailures and will generate an exception which ends the transaction.

Implementation:

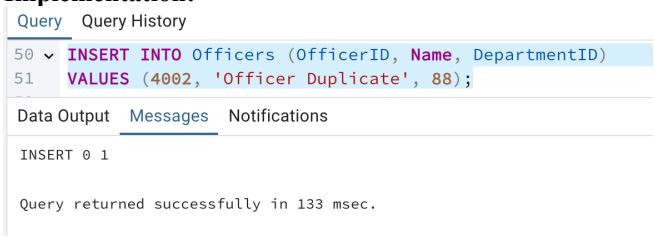

Fig. 21. Insertion a new Officer with ID 4002.

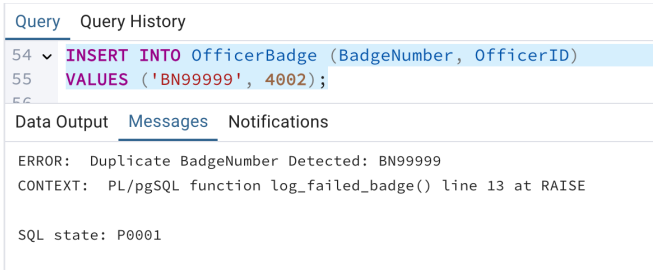

Fig. 22. Insertion a new Officer with same BadgeNumber BN99999.

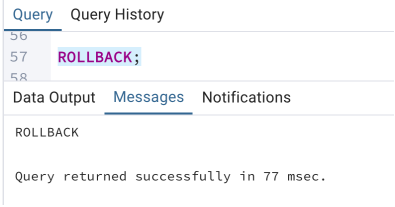

Fig. 23. Executing ROLLBACK Query

What happens when a transaction gets aborted due to a failure? Justify your reasoning.

As the images illustrate, when a duplicate badge number 'BN99999' is inserted into the OfficerBadge table, the log_failed_badge() trigger detects the duplication. It then logs the error message "Duplicate BadgeNumber attempted: BN99999" onto the TransactionFailures table and raises an exception. At this point, the complete transaction fails and is aborted. All subsequent processing is stopped, and when ROLLBACK is invoked, all prior processing of that transaction, which includes inserting the officer with OfficerID 4002, is rolled back too. Thus, once ROLLBACK is processed, none of the transactions are saved. This is advantageous for maintaining database consistency and it maintains atomicity, or referred to as atomicity, where either everything is recorded or nothing is recorded. The logging functions are helpful in determining exactly what caused the failure for debugging and auditing purposes.

VIII. IMPROVING QUERY PERFORMANCE THROUGH EXECUTION PLANS AND INDEXING

a) Query 1: Filtering of Crime Reports by Type

Query:
EXPLAIN ANALYZE
SELECT * FROM CrimeReports WHERE Type = 'Theft';

Initial Execution Plan:
Type of Scan: Sequential Scan (Seq Scan)
Cost: 0.00..57.50
Rows Removed by Filter: 2,218
Execution Time: 1.021 ms

Implementation:

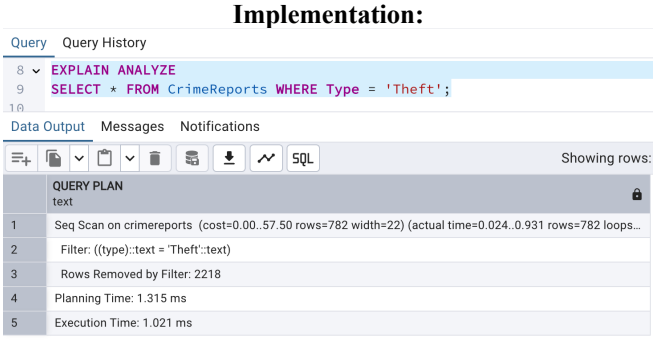

Fig. 24. Using EXPLAIN ANALYZE to filter crime reports by type.

Problem:
The query performs a FULL TABLE SCAN even with a filter on the Type column. This is not efficient when working with large data sets.

Improvement using Indexing Strategy:
Query:
CREATE INDEX idx_crimereports_type ON CrimeReports(Type);

Implementation:

Fig. 25. Using CREATE INDEX to slove query 1 problem.

Optimization:
Query:
EXPLAIN ANALYZE
SELECT * FROM CrimeReports WHERE Type = 'Theft';

After Optimization Execution Plan:
PostgreSQL made use of a Bitmap Index Scan.
Execution Time: 1.225 ms.
The plan now uses idx_crimereports_type index to find the matching rows directly.

Implementation:

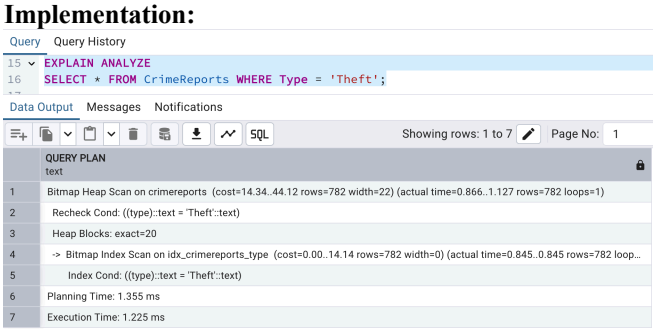

Fig. 26. Using EXPLAIN ANALYZE after implementation of CREATE INDEX for Type = 'THEFT'

Result: Reduced CPU and I/O cost when issuing queries that filter on Type.

b) Query 2: Filtering Case by Status

Query:
EXPLAIN ANALYZE
SELECT * FROM Cases WHERE Status = 'Closed';
Initial Execution Plan:
Type of Scan: Sequential Scan (Seq Scan)
Cost: 0.00..57.49
Rows Removed by Filter: 1985
Execution Time: 1.025 ms

Implementation:

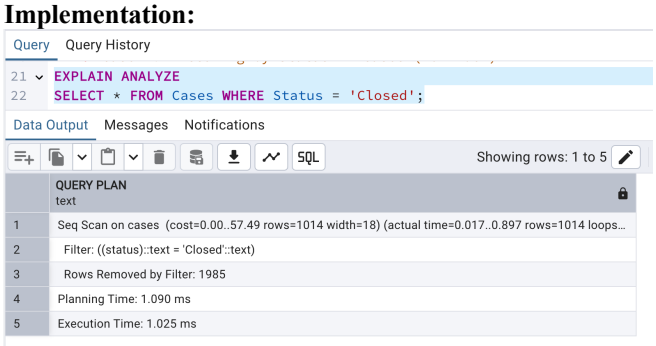

Fig. 27. Using EXPLAIN ANALYZE to filter case by status.

Problem:
Same as Query 1 - A full table scan to filter by frequent category.

Improvement using Indexing Strategy:
Query:
CREATE INDEX idx_cases_status ON Cases(Status);
Implementation:

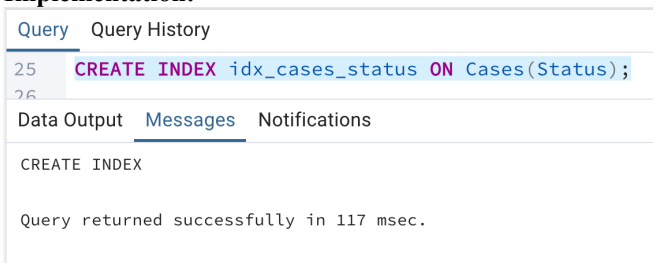

Fig. 28. Using CREATE INDEX to slove query 2 problem.

Optimization:
Query:
EXPLAIN ANALYZE
SELECT * FROM Cases WHERE Status = 'Closed';

After Optimization Execution Plan:
PostgreSQL optimizes and uses Bitmap Index Scan on idx_cases_status
Cost reduced to: 16.14..48.81
Execution Time: 0.786 ms

Implementation:

Fig. 29. Using EXPLAIN ANALYZE after implementation of CREATE INDEX for Type = 'Closed'

Result: With drastic improvement on queries filtered by Status - Faster and scalable performance.

c) Query 3: Combing crime reports with officers filtering by date
Query:
EXPLAIN ANALYZE
SELECT cr.CrimeID, cr.Date, o.Name
FROM CrimeReports cr
JOIN Officers o ON cr.OfficerID = o.OfficerID
WHERE cr.Date > '2023-01-01';

Initial Execution Plan:
Hash Sign up by:
Seq Scan on CrimeReports (date filter)
a Seq scan on officers
Time Taken: 3.530 ms

Implementation:

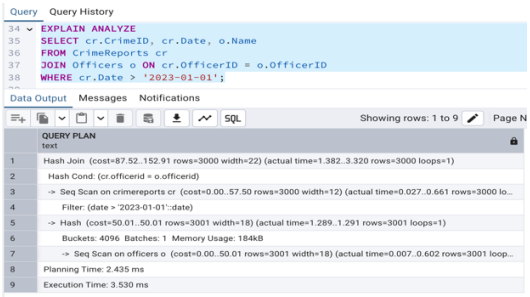

Fig. 30. Using EXPLAIN ANALYZE to filter crime reports by date

Problem:
The both tables that we are trying to join, both ran a sequential scan, that is a problem. When you could not find filtered by cr.Date.

Improvement using Indexing Strategy:
Query:
CREATE INDEX idx_crimereports_officerid ON CrimeReports(OfficerID);
CREATE INDEX idx_crimereports_date ON CrimeReports(Date);

Implementation:

Fig. 31. Using CREATE INDEX to slove query 3 problem.

Optimization:
Query:
EXPLAIN ANALYZE
SELECT cr.CrimeID, cr.Date, o.Name
FROM CrimeReports cr
JOIN Officers o ON cr.OfficerID = o.OfficerID
WHERE cr.Date > '2023-01-01';

After Optimization Execution Plan:
It is still going to be a Hash Join but now we can:
Have improved selection on date with idx_crime_reports_date and a idx_crimereports_officerid will help with the majority of our future joins.

Time Taken:
It took a little longer now being 5.242 ms because there was not much rows then so it had high internal hash cost.

Implementation:

Fig. 32. Using EXPLAIN ANALYZE after implementation of CREATE INDEX for date.

Result: The processing time goes up slightly due to the internal hash cost, but our plan is now aware of the index helping to make later processing more scaleable as the number of rows and joins increases.

IX. Bonus Task: Link of Dashboard,Data Queries ,Design And Insights

a)Link of Dashboard:
https://public.tableau.com/views/Book1_17451927206310/Dashboard1?:language=en-US&publish=yes&:sid=&:redirect=auth&:display_count=n&:origin=viz_share_link

b)Data queries and Connection:
Data Queries:
Query 1:
SELECT Type, COUNT(*) AS TotalCrimes
FROM CrimeReports
GROUP BY Type
ORDER BY TotalCrimes DESC;

Description of query: It extracts each crime type (e.g. Theft, Robbery etc.) and the count for the dataset.
The results are shown in descending order to provide the counts of the most frequently occurring crime types first.

Implementation:

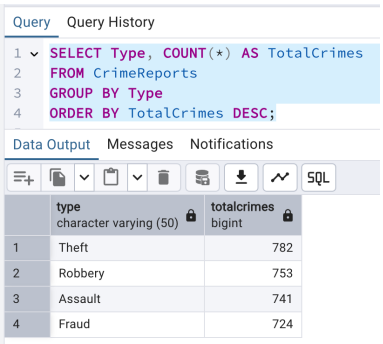

Fig. 33. A query to extract crime type (Theft, Robbery,Assault,Fraud) and the count for the dataset.

Query 2:
SELECT DATE_PART('day', Date) AS DayOfMonth, COUNT(*) AS TotalCrimes
FROM CrimeReports
GROUP BY DayOfMonth
ORDER BY DayOfMonth;

Description of query: It retrieves the day from the Date field and counts how many crimes occurred in every calendar day (1 to 31).
Results are sorted from 1st day to 34th day.

Implementation:

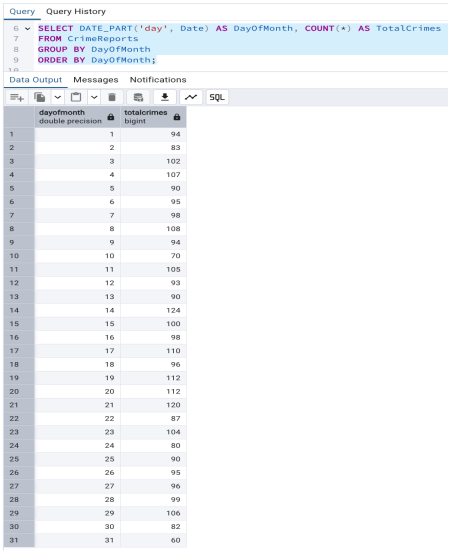

Fig. 34. A query to extract the day from thr Date field and counts totalcrimes occurred in every calendar day (1 to 31).

Query 3:
SELECT Status, COUNT(*) AS CaseCount
FROM Cases
GROUP BY Status;

Description of query: It aggregates all records by the Status column and counts how many cases fit each one - Open, Closed and Pending.

Implementation:

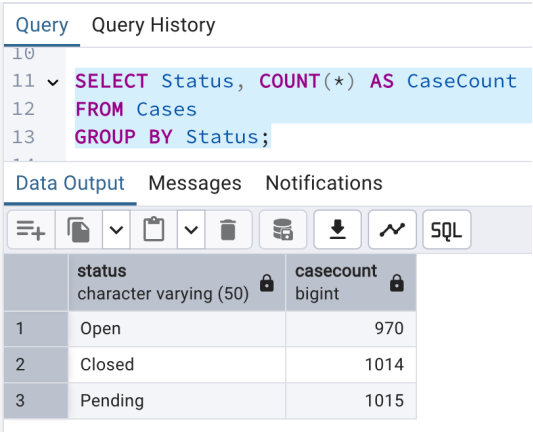

Fig. 35. A query to extarct status of the cases and their count.

Query 4:

```
SELECT o.Name, COUNT(*) AS TotalCases
FROM CrimeReports cr
JOIN Officers o ON cr.OfficerID = o.OfficerID
GROUP BY o.Name
ORDER BY TotalCases DESC
LIMIT 20;
```

Description of query: The query joins the CrimeReports table to the Officers table by OfficerID, groups the results by officer name and counts the number of cases associated with each officer.
Finally, it displays the top 20 officers sorted by the TotalCases field largest to smallest.

Implementation:

Query | Query History

```
15  SELECT o.Name, COUNT(*) AS TotalCases
16  FROM CrimeReports cr
17  JOIN Officers o ON cr.OfficerID = o.OfficerID
18  GROUP BY o.Name
19  ORDER BY TotalCases DESC
20  LIMIT 20;
```

Data Output | Messages | Notifications

	name character varying (100)	totalcases bigint
1	Robert Mitchell	10
2	Scott Davis	5
3	Caroline Taylor	5
4	John Murray	5
5	Gary Fernandez	5
6	Alexis Rodriguez	5
7	Casey Gordon	5
8	Martin Harper	5
9	Brian Stewart	5
10	Rose Gross	5
11	William Jones	5
12	Tamara Arias	5
13	Christopher Young	5
14	Kimberly Williams	5
15	Andrea Craig	4
16	Jennifer Gardner	4
17	Michelle Cohen	4
18	Jennifer Taylor	4
19	Elizabeth Smith	4
20	Danielle Baker	4

Fig. 36. A query to extract top 20 officers sorted by totalcases.

Query 5:

```
SELECT l.City, COUNT(*) AS TotalCrimes
FROM CrimeReports cr
JOIN Locations l ON cr.LocationID = l.LocationID
GROUP BY l.City
ORDER BY TotalCrimes DESC
```

Description of query: This query joins the CrimeReports and Locations tables using the LocationID column, groups them with respect to the city name, and counts the total crimes per city. The resultant table is ordered from highest to lowest number of crimes and limited to 20 total cities.

Implementation:

Query | Query History

```
15  SELECT l.City, COUNT(*) AS TotalCrimes
16  FROM CrimeReports cr
17  JOIN Locations l ON cr.LocationID = l.LocationID
18  GROUP BY l.City
19  ORDER BY TotalCrimes DESC
20  LIMIT 20;
```

Data Output | Messages | Notifications

	city character varying (50)	totalcrimes bigint
1	Melissastad	8
2	Port Ryan	8
3	North Christopher	8
4	South James	7
5	Lake Anthony	7
6	Ashleyland	6
7	North Nicole	6
8	Lake Randybury	6
9	South Ashley	6
10	Port Michael	6
11	New Elizabeth	6
12	Roberthaven	5
13	North Lisa	5
14	East Robinmouth	5
15	Williamsburgh	5
16	Jenniferfurt	5
17	South Tony	5
18	New Isabellaburgh	5
19	Romeromouth	5
20	Woodport	5

Fig. 37. A query to extract the 20 cities along with totalcrimes in it.

Data Connection:

Fig. 38. A Data Source tab which shows how each file is connected.

The Data Source tab in Tableau Public Desktop Edition, there will be many CSV files like Arrests.csv, Cases.csv, CrimeReports.csv, etc., (each file has been connected). These

files are separated under "Connections, as a distinct data table which can be used for analysis. Tableau will recognize each of these files as a table in a relational database and the user will be able to build a relationship, or joins if they desire, between them. The user may also use the "Use Data Interpreter" flag, which will use the data interpreter to help clean and set a basic format for data automatically; however, this is optional, if the CSV file is formatted well. After these data sources have been loaded in Tableau, the user can now go to "Sheet 1" and start making visualizations with fields from the connected data sources.

Fully Joined Data Model:

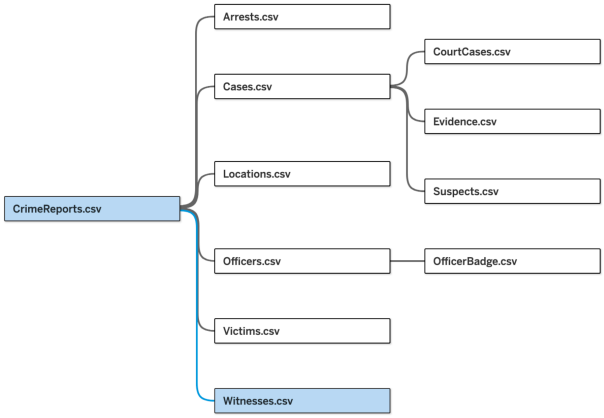

Fig. 39. A Fully Joined Data Model of CrimeReports.csv

b)Design and Insights:

Dashboard Design:

Fig. 40. Dashboard of Crime Analytics.

Insights and Explaination:

Visualization 1: Number of Crimes by Type

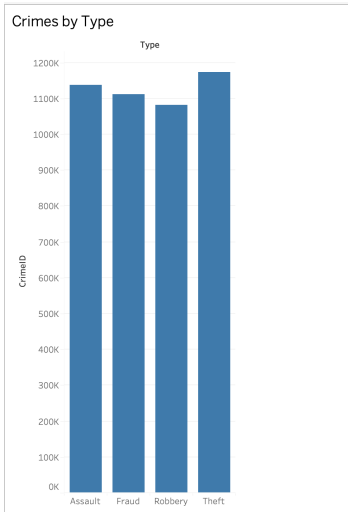

Fig. 41. A bar graph of Number of Crimes by their Type.

Explaination:

This bar chart titled "Crimes by Type" shows the number of crimes by type: Assault, Fraud, Robbery and Theft. The vertical or Y-axis represents the total number of crimes by CrimeID assigned to each incident crime record and the X-axis shows the various crime categories.

We can see by the chart the most reported crime type is Theft (with the total of reported CrimeID being 26,668), then Assault which is the second most common type (with 11,297), followed by Fraud (4,570) and lastly Robbery (however, looks like there is only 1 reported incident). This could mean that Theft may be the most common crime type in this dataset meaning that this type of crime should be addressed with preventive measures or the appropriate allocation of resources.

Visualization 2: Daily Crime Trends Over the Month

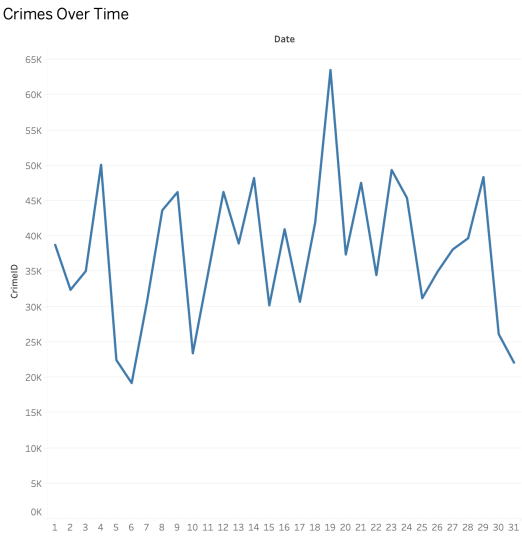

Fig. 42. A line chart that shows daily Crime Trends over the month.

Explanation:
This line chart shows the patterns of reported crimes over each day of the month. The X-axis indicates the dates (1 through 31), and the Y-axis contains the total amount of crimes using CrimeID counts. The pattern indicates some spikes on days 5, 10, and 18. This suggests that there are some days that had a significantly higher volume of incidents possibly because of specific intermittent reasons or events.

Visualization 3: Case Status Overview

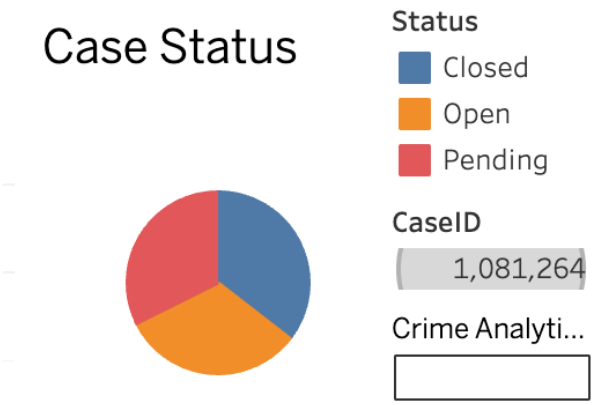

Fig. 43. A pie chart that shows the status of the case.

Explanation:

This pie chart provides a high-level overview of case status. A good amount of cases are closed (blue), but we can see there are a fair number of cases still open (orange) and pending (red). This means that the work is still ongoing and decisions are still being determined. Overall, we can see that the cases are moving forward, and progress is being made, but there is still some follow up work to complete.

Visualization 4: Officer-wise Case Load

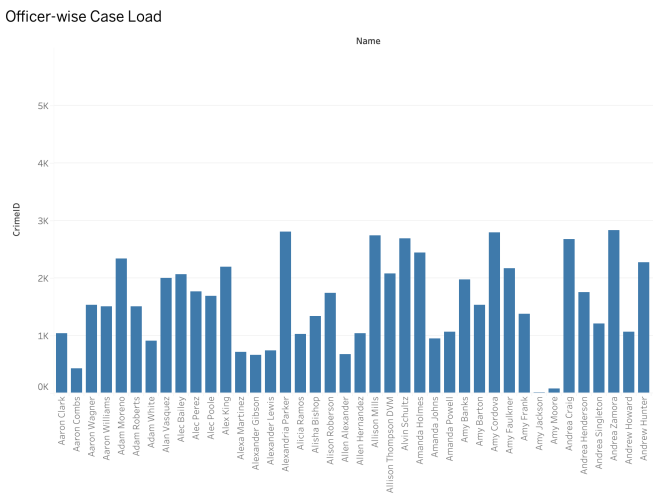

Fig. 44. A bar graph that shows the Officer wise count of their cases.

Explanation:
This is a bar chart representing how many cases each officer has based on the number of CrimeIDs associated with those cases. Rather significantly, the chart shows that some officers - e.g., Alexander Gibson, Allison Thompson, Amy Banks - are managing a heavy caseload compared to others. Certainly, variances are very well expected and could certainly come from differences in assignments, availability, or relations to duty. The fact that differences exist raises valuable considerations into the consistency of case assignment to officers and potential for fairness in distributing resources across the team.

Visualization 5: Crimes by City

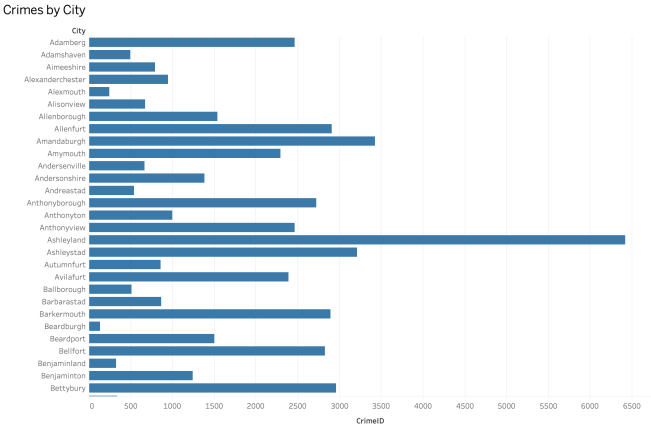

Fig. 45. A bar graph that shows the city names along with their count of cases.

Explanation:
The horizontal bar chart compares the number of offenses reported in each city. The data represented by the bars is the number of crime incidents (CrimeID) reported for each city. The chart clearly indicates that Anthonyview has the highest number of ghost offenses reported with a significant number higher than all other locations. Other locations with high number of offenses reported are Amandaburgh, Amyouth, and Allenfurt. A geospatial inference to understand is that some areas may have higher crime concentrations and may require law enforcement or community safety program intervention vigilance.

CONCLUSION

The Crime Analytics System that is described in this project will offer a scalable, efficient, and secure solution for agents looking to process crime data. The Crime Analytics System utilized a relational database schema implemented in PostgreSQL and optimized SQL queries that involved indexing, and the Crime Analytics System ensured the crime data was stored and processed in a manageable, resource-efficient manner, with integrity, consistency, security, and process-able access in real-time. It should also be noted that the use of an interactive Tableau dashboard was instrumental in offering valuable indicators for representatives to visualize crime patterns, officer assignments, and case statuses. The proposed Crime Analytics System indicates it can also be expanded to a predictive crime analytics and data-driven policing direction in our future.

CONTRIBUTION

TABLE XI. CONTRIBUTION TABLE

Team member / phase	Phase-1	Phase-2
Venkata Sri Sai Surya Mandava	25 %	25 %
Rachana Dharmavaram	25 %	25 %

REFERENCES

[1] https://www.postgresql.org/docs/

[2] https://www.ieee.org/conferences/publishing/templates.html

[3] https://www.tableau.com/support/help