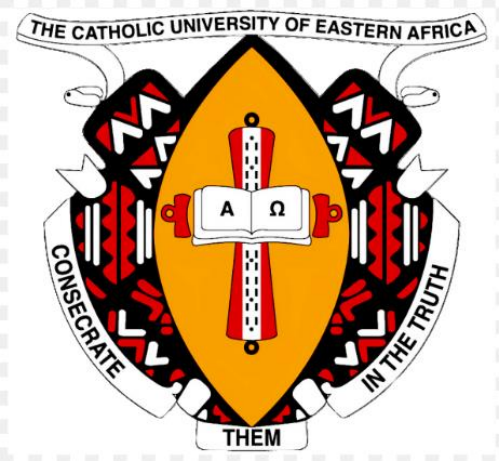


THE CATHOLIC UNIVERSITY OF EASTERN AFRICA (CUEA)

CMT302: ADVANCED DATABASE SYSTEMS

GROUP PROJECT (CAT 2)

TRAFFIC VIOLATION MANAGEMENT SYSTEM

GROUP 20

Group members:

Patience Ndanu – 1050834

Adhiambo Laura Loreto – 1049079

Chantel Gathunguri - 1050820

Ziri Ruth Mwake – 1052019

Marietta Mbithe – 1050894

Submitted, 19th November 2024

Table of Contents

OVERVIEW

The Traffic Violation Management System is an advanced database-driven solution designed to manage traffic violations and penalties. It efficiently integrates various tables and functionalities, handling key entities such as drivers, vehicles, violations and law enforcement officers, therefore creating a centralized platform for all these processes. The system reduces the dependency on manual processes since it automates key functions like the recording of violations, generation of violation reports and computation of fines. It also integrates triggers that automate actions such as violation report creation once a new violation is logged and stored procedures to handle repetitive tasks in the system.

In addition, the system includes advanced reports that enable the analysis of traffic violation patterns, assessment of fine collections and evaluation of officer performance. This information is crucial in improving traffic management while enhancing public safety. By having a digital traffic violation management system, we can enhance the efficiency and accountability of traffic law enforcement and ultimately contribute to safer road environments.

OBJECTIVES

Data Storage - To maintain detailed, inter-related records about drivers, vehicles, officers, and violation reports.

Smoothened Operations- Ensuring efficient data entry, updates and retrieval through the use of automated CRUD operations (Create, Read, Update and Delete)

Better Decision Making- Use advanced SQL reporting and analytics to identify emerging trends, evaluate officer performance and analyze collection of fines

Improved compliance and enforcement- Ensure timely penalty enforcement with real time insights into outstanding fines and violations.

User accessibility and security- This will enable users to access the system through the use of a secure interface while granting different levels of access and encryption

Future Scalability- More development including optimization for large data sets and integration with devices such as traffic cameras to automate the violation detection and logging processes.

RATIONALE

When ensuring road safety and law enforcement, managing traffic violations is a critical aspect. Traditional methods where data entry is manual are prone to a number of inconveniences such as errors, delays and inefficiencies. They can also prove to be time-consuming. The traffic violation management system addresses these challenges with its features where it implements a well-structured database system that simplifies operations and enhances law enforcement.

Since the management of violations is automated, the processing of penalties is sped up and accurate records are maintained because human error is minimized. The system integrates features such as triggers, stored procedures, and advanced SQL queries which provide a foundation for automatic violation reports, fine adjustments based on the type of violation and comprehensive reports on officer performance and violation patterns which are the key capabilities of the system. Future scalability of the system can even offer support to modern needs such as violation

notifications via SMS or integration with IoT devices which are valuable features in our growing digital age.

Fundamentally, by adopting traffic violating management systems, greater compliance with traffic laws can be ensured while improving resource allocations and thereby providing a safe, transparent and accountable environment for all road users.

SYSTEM DESIGNS

SQL schema

1. **Driver**
    i. DriverID (Primary Key)
    ii. Name
    iii. LicenseNumber (Unique)
    iv. Address
    v. PhoneNumber
2. **Vehicle**
    i. VehicleID (Primary Key)
    ii. LicensePlate (Unique)
    iii. Make
    iv. Model
    v. Year
    vi. OwnerID (Foreign Key referencing DriverID)

3. **TrafficViolation**

    i. ViolationID (Primary Key)

    ii. ViolationType

    iii. ViolationDate

    iv. Location

    v. FineAmount

    vi. DriverID (Foreign Key referencing DriverID)

    vii. VehicleID (Foreign Key referencing VehicleID)

4. **Officer**

    i. OfficerID (Primary Key)

    ii. Name

    iii. BadgeNumber (Unique)

5. **ViolationReport**

    i. ReportID (Primary Key)

    ii. ViolationID (Foreign Key referencing ViolationID)

    iii. OfficerID (Foreign Key referencing OfficerID)

    iv. IssuedDate

Rockoff, L. (2021).

Relationships:

1. A Driver can own multiple Vehicles.
2. A Driver can have multiple Traffic Violations.
3. A Traffic Violation can be issued by one Officer.
4. An Officer can issue multiple Violation Reports for different Traffic Violations.

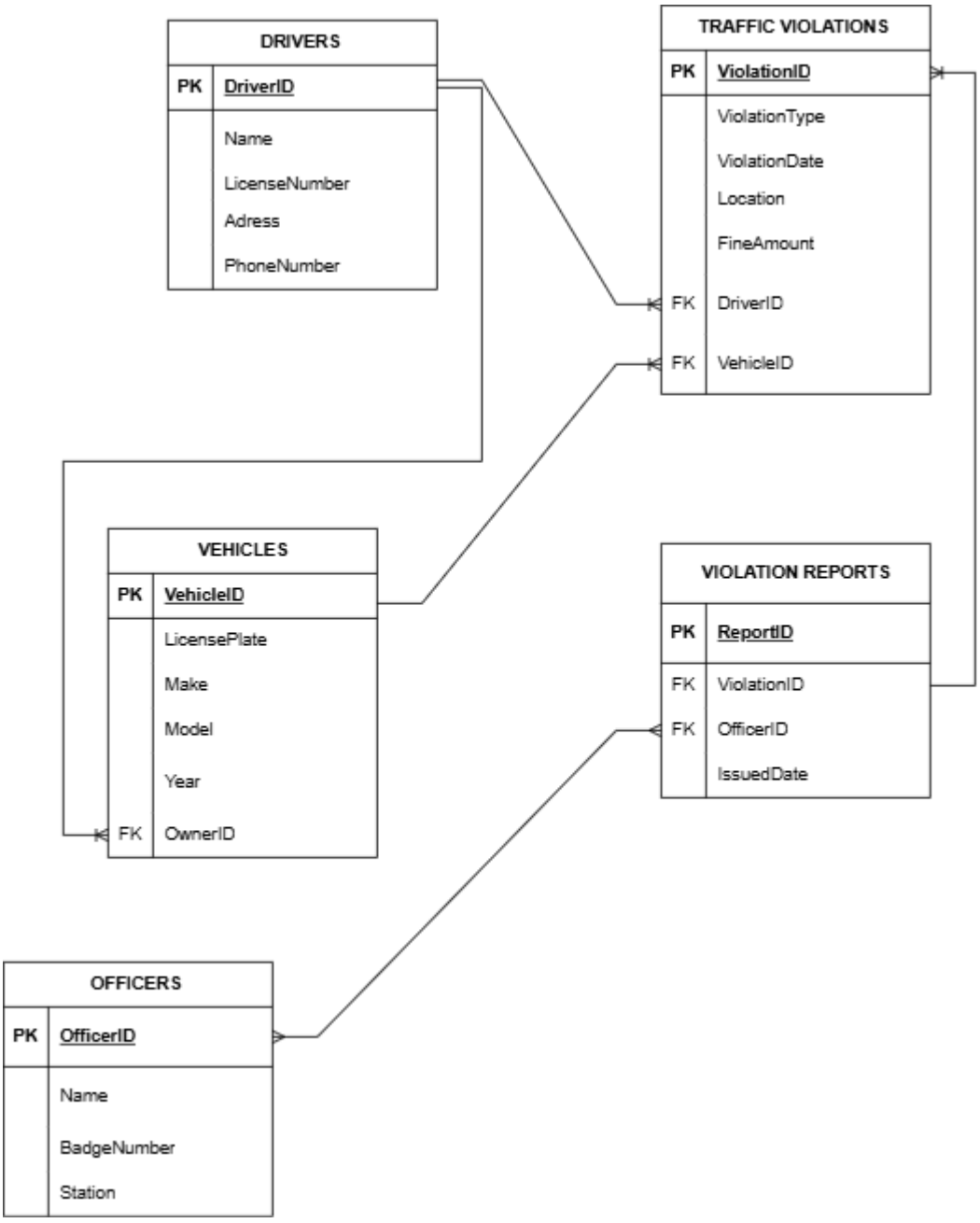
Table creation scripts

```
CREATE TABLE Driver (
    DriverID INT PRIMARY KEY,
    Name VARCHAR(100),
    LicenseNumber VARCHAR(50) UNIQUE,
    Address VARCHAR(255),
    PhoneNumber VARCHAR(15)
);

CREATE TABLE Vehicle (
    VehicleID INT PRIMARY KEY,
```

```
    LicensePlate VARCHAR(15) UNIQUE,
    Make VARCHAR(50),
    Model VARCHAR(50),
    Year INT,
    OwnerID INT,
    FOREIGN KEY (OwnerID) REFERENCES Driver(DriverID)
);

CREATE TABLE TrafficViolation (
    ViolationID INT PRIMARY KEY,
    ViolationType VARCHAR(100),
    ViolationDate DATE,
    Location VARCHAR(255),
    FineAmount DECIMAL(10, 2),
    DriverID INT,
    VehicleID INT,
    FOREIGN KEY (DriverID) REFERENCES Driver(DriverID),
    FOREIGN KEY (VehicleID) REFERENCES Vehicle(VehicleID)
```

```
);

CREATE TABLE Officer (
    OfficerID INT PRIMARY KEY,
    Name VARCHAR(100),
    BadgeNumber VARCHAR(50) UNIQUE,
    Station VARCHAR(100)
);

CREATE TABLE ViolationReport (
    ReportID INT PRIMARY KEY,
    ViolationID INT,
    OfficerID INT,
    IssuedDate DATE,
    FOREIGN KEY (ViolationID) REFERENCES TrafficViolation(ViolationID),
    FOREIGN KEY (OfficerID) REFERENCES Officer(OfficerID)
);
```

ER Diagram


DRIVERS
PK
DriverID
Name
LicenseNumber
Adress
PhoneNumber
TRAFFIC VIOLATIONS
PK
ViolationID
ViolationType
ViolationDate
Location
FineAmount
FK
DriverID
FK
VehicleID
VEHICLES
PK
VehicleID
LicensePlate
Make
Model
Year
FK
OwnerID
VIOLATION REPORTS
PK
ReportID
FK
ViolationID
FK
OfficerID
IssuedDate
OFFICERS
PK
OfficerID
Name
BadgeNumber
Station

IMPLEMENTATION

CRUD OPERATIONS

1. Driver Table

INSERT INTO Driver (DriverID, Name, LicenseNumber, Address, PhoneNumber)

VALUES (?, ?, ?, ?, ?);

SELECT * FROM Driver WHERE DriverID = ?;

UPDATE Driver SET Name = ?, LicenseNumber = ?, Address = ?, PhoneNumber = ?

WHERE DriverID = ?;

DELETE FROM Driver WHERE DriverID = ?;

2. Vehicle Table

INSERT INTO Vehicle (VehicleID, LicensePlate, Make, Model, Year, OwnerID)

VALUES (?, ?, ?, ?, ?, ?);

SELECT * FROM Vehicle WHERE VehicleID = ?;

UPDATE Vehicle SET LicensePlate = ?, Make = ?, Model = ?, Year = ?, OwnerID = ?

WHERE VehicleID = ?;

DELETE FROM Vehicle WHERE VehicleID = ?;

3. **TrafficViolation Table**

INSERT INTO TrafficViolation (ViolationID, ViolationType, ViolationDate, Location, FineAmount, DriverID, VehicleID)

VALUES (?, ?, ?, ?, ?, ?, ?);

SELECT * FROM TrafficViolation WHERE ViolationID = ?;

UPDATE TrafficViolation SET ViolationType = ?, ViolationDate = ?, Location = ?, FineAmount = ?, DriverID = ?, VehicleID = ?

WHERE ViolationID = ?;

DELETE FROM TrafficViolation WHERE ViolationID = ?;

4. **Officer Table**

INSERT INTO Officer (OfficerID, Name, BadgeNumber, Station)

VALUES (?, ?, ?, ?);

SELECT * FROM Officer WHERE OfficerID = ?;

UPDATE Officer SET Name = ?, BadgeNumber = ?, Station= ?

WHERE OfficerID = ?;

DELETE FROM Officer WHERE OfficerID = ?;

5. **ViolationReport Table**

INSERT INTO ViolationReport (ReportID, ViolationID, OfficerID, IssuedDate)

VALUES (?, ?, ?, ?);

SELECT * FROM ViolationReport WHERE ReportID = ?;

UPDATE ViolationReport SET ViolationID = ?, OfficerID = ?, IssuedDate = ?

WHERE ReportID = ?;

DELETE FROM ViolationReport WHERE ReportID = ?;

STORED PRECEDURES

1. **Add a new Driver**

DELIMITER //

CREATE PROCEDURE AddDriver(

```
    IN p_DriverID INT,
    IN p_Name VARCHAR(100),
    IN p_LicenseNumber VARCHAR(50),
    IN p_Address VARCHAR(255),
    IN p_PhoneNumber VARCHAR(15)
)
BEGIN
    INSERT INTO Driver (DriverID, Name, LicenseNumber, Address, PhoneNumber)
    VALUES (p_DriverID, p_Name, p_LicenseNumber, p_Address, p_PhoneNumber);
END //
DELIMITER;
```

2. **Update a traffic violation fine**

```
DELIMITER //
CREATE PROCEDURE UpdateViolationFine(
    IN p_ViolationID INT,
    IN p_FineAmount DECIMAL(10, 2)
)
```

```
BEGIN
    UPDATE TrafficViolation
    SET FineAmount = p_FineAmount
    WHERE ViolationID = p_ViolationID;
END //
DELIMITER ;
```

3. **Get violations for a driver**

```
DELIMITER //
CREATE PROCEDURE GetDriverViolations(
    IN p_DriverID INT
)
BEGIN
    SELECT * FROM TrafficViolation
    WHERE DriverID = p_DriverID;
END //
DELIMITER ;
```

TRIGGERS

1. **Trigger to insert a report when a violation is added;**

```
DELIMITER //

CREATE TRIGGER AfterTrafficViolationInsert

AFTER INSERT ON TrafficViolation

FOR EACH ROW

BEGIN

   DECLARE p_OfficerID INT DEFAULT 1;  -- Example: Default officer ID

   INSERT INTO ViolationReport (ViolationID, OfficerID, IssuedDate)

   VALUES (NEW.ViolationID, p_OfficerID, NOW());

END //

DELIMITER ;
```

2. **Trigger to log any deletions made to the Driver Table into a separate log table;**

```
DELIMITER //

CREATE TABLE DriverDeletionLog (

   LogID INT AUTO_INCREMENT PRIMARY KEY,

   DeletedDriverID INT,

   DeletionDate DATETIME
```

```
);


CREATE TRIGGER AfterDriverDelete
AFTER DELETE ON Driver
FOR EACH ROW
BEGIN
    INSERT INTO DriverDeletionLog (DeletedDriverID, DeletionDate)
    VALUES (OLD.DriverID, NOW());
END //
DELIMITER ;
```

3. **Trigger to automatically adjust the FineAmount based on the ViolationType if updated;**

```
DELIMITER //
CREATE TRIGGER BeforeViolationUpdate
BEFORE UPDATE ON TrafficViolation
FOR EACH ROW
BEGIN
    IF NEW.ViolationType != OLD.ViolationType THEN
```

```
        SET NEW.FineAmount = CASE
            WHEN NEW.ViolationType = 'Speeding' THEN 100.00
            WHEN NEW.ViolationType = 'Running a Red Light' THEN 200.00
            ELSE 50.00
        END;
    END IF;
END //
DELIMITER ;
```

ADVANCED SQL REPORTS

1. **Combines data from multiple tables.**

   To get a list of violations, including driver and officer details.

```
SELECT
    tv.ViolationID,
    tv.ViolationType,
    tv.ViolationDate,
    d.Name AS DriverName,
    o.Name AS OfficerName,
    tv.FineAmount
FROM
```

```
    TrafficViolation tv
JOIN Driver d ON tv.DriverID = d.DriverID
JOIN ViolationReport vr ON tv.ViolationID = vr.ViolationID
JOIN Officer o ON vr.OfficerID = o.OfficerID;
```

2. **Aggregation**

To summarize the total fines collected by each officer.

```
SELECT
    o.Name AS OfficerName,
    SUM(tv.FineAmount) AS TotalFinesCollected
FROM
    TrafficViolation tv
JOIN ViolationReport vr ON tv.ViolationID = vr.ViolationID
JOIN Officer o ON vr.OfficerID = o.OfficerID
GROUP BY o.Name
ORDER BY TotalFinesCollected DESC;
```

3. **Subqueries**

To find drivers with the highest number of violations.

```
SELECT
    d.Name AS DriverName,
```

```
    COUNT(tv.ViolationID) AS ViolationCount
FROM
    Driver d
JOIN TrafficViolation tv ON d.DriverID = tv.DriverID
GROUP BY d.Name
HAVING ViolationCount = (
    SELECT MAX(ViolationCount)
    FROM (
        SELECT COUNT(ViolationID) AS ViolationCount
        FROM TrafficViolation
        GROUP BY DriverID
    ) AS SubQuery
);
```

4. **Window functions (Rankings and trends)**

To rank violations by the fine amount.

```
SELECT
    tv.ViolationID,
    tv.ViolationType,
    tv.FineAmount,
    RANK() OVER (ORDER BY tv.FineAmount DESC) AS RankByFine
FROM
```

TrafficViolation tv;

5. **Conditional statements**

To categorize fines based on their amounts.

```
SELECT
    tv.ViolationID,
    tv.ViolationType,
    tv.FineAmount,
    CASE
        WHEN tv.FineAmount < 100 THEN 'Low'
        WHEN tv.FineAmount BETWEEN 100 AND 500 THEN 'Medium'
        ELSE 'High'
    END AS FineCategory
FROM
    TrafficViolation tv;
```

GENERATING REPORTS

1. **Monthly fines collected**

```
    SELECT
```

```
    DATE_FORMAT(tv.ViolationDate, '%Y-%m') AS Month,
    SUM(tv.FineAmount) AS TotalFines
FROM
    TrafficViolation tv
GROUP BY DATE_FORMAT(tv.ViolationDate, '%Y-%m')
ORDER BY Month;
```

2. **Drivers with most violations**

```
SELECT
    d.Name AS DriverName,
    COUNT(tv.ViolationID) AS TotalViolations
FROM
    Driver d
JOIN TrafficViolation tv ON d.DriverID = tv.DriverID
GROUP BY d.DriverID, d.Name
ORDER BY TotalViolations DESC
LIMIT 10;
```

3. **Violations by location**

```
SELECT
    tv.Location,
    COUNT(tv.ViolationID) AS TotalViolations,
    SUM(tv.FineAmount) AS TotalFines
FROM
    TrafficViolation tv
GROUP BY tv.Location
ORDER BY TotalViolations DESC;
```

4. **Officer performance**

```
SELECT
    o.Name AS OfficerName,
    COUNT(vr.ReportID) AS ViolationsReported,
    SUM(tv.FineAmount) AS TotalFinesCollected
FROM
    Officer o
LEFT JOIN ViolationReport vr ON o.OfficerID = vr.OfficerID
LEFT JOIN TrafficViolation tv ON vr.ViolationID = tv.ViolationID
GROUP BY o.OfficerID, o.Name
ORDER BY ViolationsReported DESC;
```

5. **Outstanding fines**

```
SELECT
    d.Name AS DriverName,
    tv.ViolationID,
    tv.FineAmount,
    tv.ViolationDate
FROM
    TrafficViolation tv
JOIN Driver d ON tv.DriverID = d.DriverID
WHERE tv.FineAmount > 0
ORDER BY tv.ViolationDate ASC;
```

TESTING AND VALIDATION

The **'table creation scripts'** were tested in MySQL command line and it produced the expected outcome where all tables were successfully created in our TrafficViolation. (Appendix 1)

CRUD operations:

The CRUD operations were used to insert data to all tables, update the data to make any changes and delete some records in the table.

Stored Procedures:

The stored procedures scripts were tested in the command line and the expected outcome was met. All stored procedure were successfully inserted in the database. (Appendix 2, 3 and 4)

Triggers:

The Triggers scripts were tested in the command line and the expected outcome was met. All triggers under their respective tables were successfully inserted into the database. (Appendix 5 and 6)

Advanced SQL Reports and Generating Reports:

The codes to generate the various reports were tested in the command line the expected output was generated. (Appendix 7, 8, 9 and 10)

CONCLUSION AND RECOMMENDATIONS

The Traffic Violation Management System demonstrates the application of sophisticated database principles, like designing SQL schemas and performing CRUD operations and stored procedures along with triggers, in a setting. Extensive testing was conducted on the system to validate the functionality of all incorporated features ensuring handling of traffic violation data. Additionally the utilization of advanced SQL reporting capabilities augments the systems capacity to produce analyses including fines evaluation, policing officer efficiency assessment and identification of violation patterns. The project showcases a hands, on and effective database

solution for overseeing traffic violations, with ease and success shown through the execution and validation of all elements involved.

Although the system meets the needs, for managing traffic violations more enhancements can be made to improve its efficiency and usability as well as scalability. There is a focus, on creating a user interface that enables officers and administrators to interact with the system without needing to use direct SQL commands. Security measures are also being beefed up with the addition of role based access controls and data encryption to protect information.

The system could also improve by including automated notification capabilities to inform drivers of infractions and penalties through email or text messages. In addition, to that feature enhancements could involve real time analytics displays offering up to date information on traffic violation patterns and officer efficiency. Furthermore integrating Internet of Things gadgets, like speed cameras or license plate scanners would simplify the task of detecting and notifying about violations automatically.

To support broader deployment, the database and queries should be optimized for handling large datasets, ensuring scalability for extensive use cases. Additionally, creating a mobile application for officers would facilitate real-time logging of violations, enhancing the system's flexibility and responsiveness. Implementing these advancements would elevate the system into a comprehensive tool for managing traffic violations, offering improved efficiency, enhanced user experience, and the ability to scale for future needs.

REFERENCES

Groff, J. R., Weinberg, P. N., & Oppel, A. J. (2002). *SQL: the complete reference* (Vol. 2). McGraw-Hill/Osborne.

Hu, Y., Jiang, H., Tang, H., Lin, X., & Hu, Z. (2023). SQL#: a language for maintainable and debuggable database queries. *International Journal of Software Engineering and Knowledge Engineering*, *33*(05), 619-649.

Li, Y., Chen, L., Yu, D., & Gao, R. (2021, June). Research and developing of evaluation information system using B/S structure and SQL server technology. In *Journal of Physics: Conference Series* (Vol. 1952, No. 4, p. 042088). IOP Publishing.

Meier, A., & Kaufmann, M. (2019). *SQL & NoSQL databases* (pp. 123-142). Wiesbaden:: Springer Fachmedien Wiesbaden.

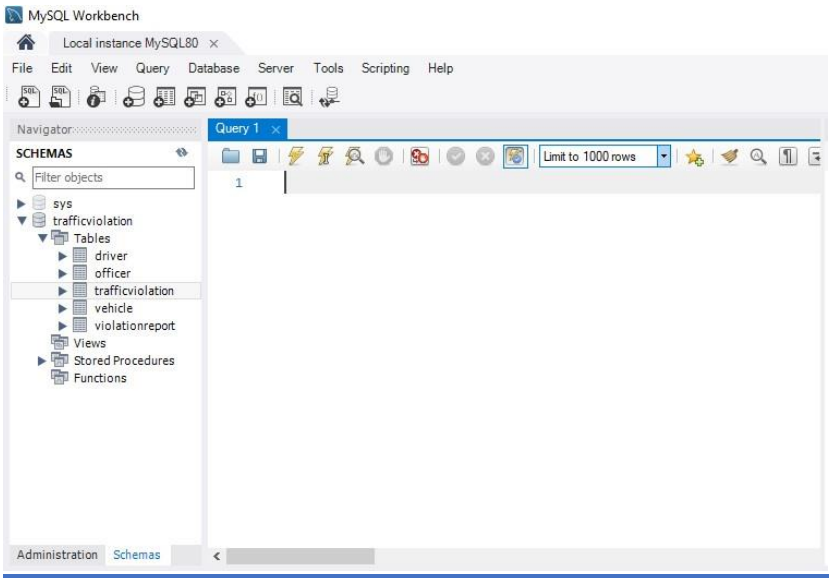
Mukherjee, S. (2019). SQL Server Development Best Practices. *International Journal of Innovative Research in Computer and Communication Engineering*, *10*.

Rockoff, L. (2021). *The language of SQL*. Addison-Wesley Professional.

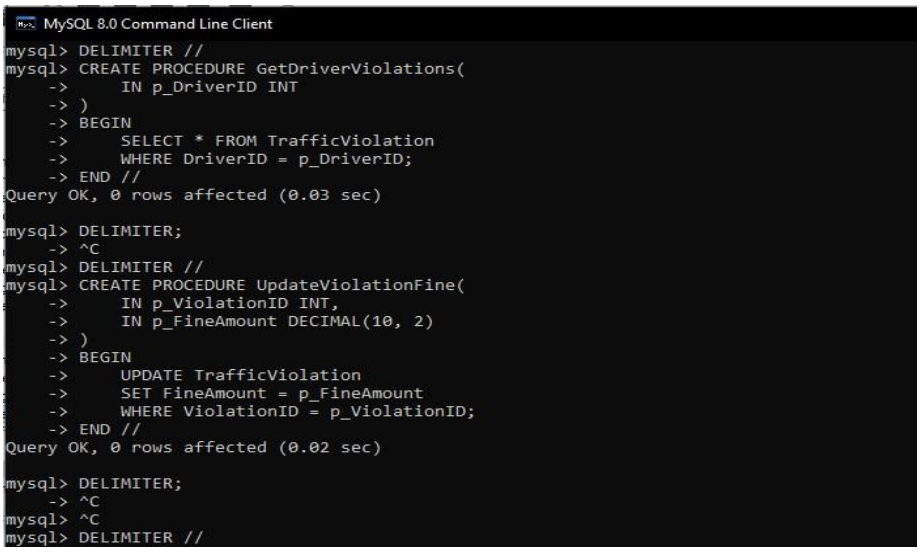
Sander, A., & Wauer, R. (2019). Integrating terminologies into standard SQL: a new approach for research on routine data. *Journal of biomedical semantics*, *10*, 1-11.

Taipalus, T., & Seppänen, V. (2020). SQL education: A systematic mapping study and future research agenda. *ACM Transactions on Computing Education (TOCE)*, *20*(3), 1-33.

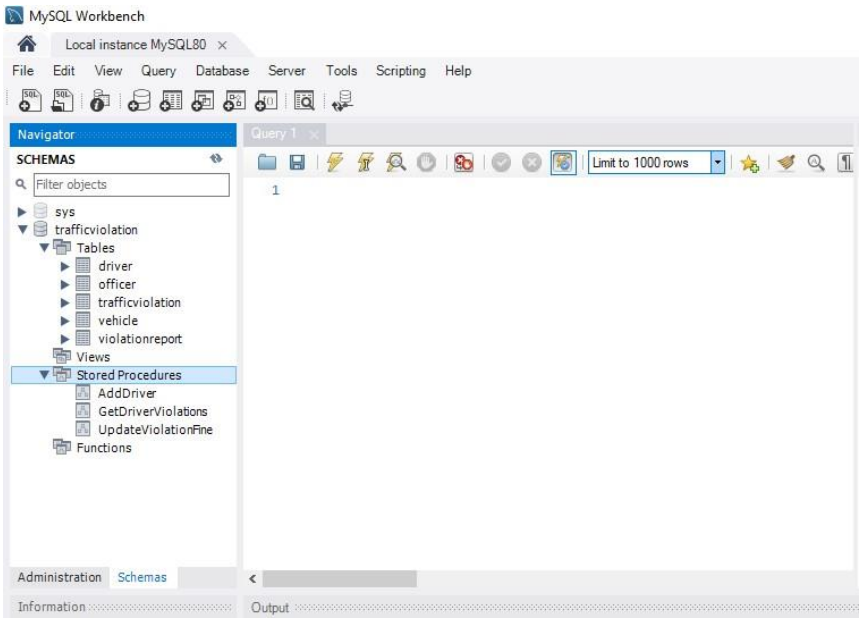
APPENDICES



Appendix 1



Appendix 2; creating the stored procedures



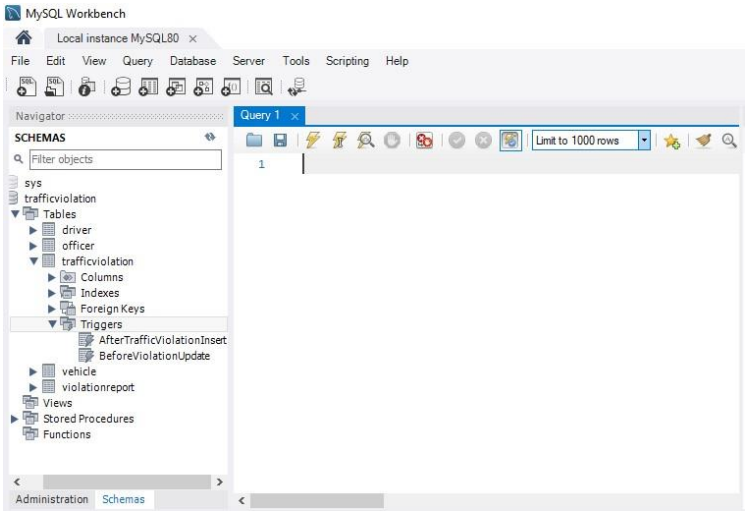
Appendix 3; shows the stored procedures in the database

```
mysql> CALL GetDriverViolations(2);
+-------------+---------------+---------------+--------------+------------+----------+-----------+
| ViolationID | ViolationType | ViolationDate | Location     | FineAmount | DriverID | VehicleID |
+-------------+---------------+---------------+--------------+------------+----------+-----------+
|           3 | Speeding      | 2024-09-01    | Langata Road |     100.00 |        2 |         3 |
+-------------+---------------+---------------+--------------+------------+----------+-----------+
1 row in set (0.01 sec)

Query OK, 0 rows affected (0.03 sec)

mysql>
```

Appendix 4; stored procedure execution of GetDriverViolations

```
MySQL 8.0 Command Line Client
mysql> DELIMITER //
mysql> CREATE TRIGGER AfterTrafficViolationInsert
    -> AFTER INSERT ON TrafficViolation
    -> FOR EACH ROW
    -> BEGIN
    ->     DECLARE p_OfficerID INT DEFAULT 1;  -- Example: Default officer ID
    ->     INSERT INTO ViolationReport (ViolationID, OfficerID, IssuedDate)
    ->     VALUES (NEW.ViolationID, p_OfficerID, NOW());
    -> END //
Query OK, 0 rows affected (0.09 sec)

mysql> DELIMITER;
    -> ^C
mysql> DELIMITER //
mysql> CREATE TRIGGER BeforeViolationUpdate
    -> BEFORE UPDATE ON TrafficViolation
    -> FOR EACH ROW
    -> BEGIN
    ->     IF NEW.ViolationType != OLD.ViolationType THEN
    ->         SET NEW.FineAmount = CASE
    ->             WHEN NEW.ViolationType = 'Speeding' THEN 100.00
    ->             WHEN NEW.ViolationType = 'Running a Red Light' THEN 200.00
    ->             ELSE 50.00
    ->         END;
    ->     END IF;
    -> END //
Query OK, 0 rows affected (0.03 sec)

mysql> DELIMITER //
mysql> DELIMITER;
    -> ^C
mysql>
```

Appendix 5; creating the triggers



Appendix 6; shows triggers in the database

```
mysql> DELIMITER ;
mysql> SELECT
    ->     tv.ViolationID,
    ->     tv.ViolationType,
    ->     tv.ViolationDate,
    ->     d.Name AS DriverName,
    ->     o.Name AS OfficerName,
    ->     tv.FineAmount
    -> FROM
    ->     TrafficViolation tv
    -> JOIN Driver d ON tv.DriverID = d.DriverID
    -> JOIN ViolationReport vr ON tv.ViolationID = vr.ViolationID
    -> JOIN Officer o ON vr.OfficerID = o.OfficerID;
+-------------+-------------------+---------------+--------------+---------------------+------------+
| ViolationID | ViolationType     | ViolationDate | DriverName   | OfficerName         | FineAmount |
+-------------+-------------------+---------------+--------------+---------------------+------------+
|           1 | Running a Red Light | 2024-10-20    | James Bond   | Officer Joe Mwangi  |     250.00 |
|           2 | Speeding          | 2024-09-11    | James Bond   | Officer Joe Mwangi  |     500.00 |
|           3 | Speeding          | 2024-09-01    | Joseph Kamau | Officer Jimmy Omondi |     100.00 |
|           4 | Careless overtaking | 2024-11-17    | Kevin        | Officer Omar Zidan  |     100.00 |
|           5 | Break Checking    | 2024-08-07    | Matthew Smith | Officer Elvis Kamau |     200.00 |
+-------------+-------------------+---------------+--------------+---------------------+------------+
5 rows in set (0.01 sec)

mysql>
```

Appendix 7 ; combine data from multiple table

```
mysql> SELECT
    ->     o.Name AS OfficerName,
    ->     SUM(tv.FineAmount) AS TotalFinesCollected
    -> FROM
    ->     TrafficViolation tv
    -> JOIN ViolationReport vr ON tv.ViolationID = vr.ViolationID
    -> JOIN Officer o ON vr.OfficerID = o.OfficerID
    -> GROUP BY o.Name
    -> ORDER BY TotalFinesCollected DESC;
+---------------------+---------------------+
| OfficerName         | TotalFinesCollected |
+---------------------+---------------------+
| Officer Joe Mwangi  |              750.00 |
| Officer Elvis Kamau |              200.00 |
| Officer Jimmy Omondi |              100.00 |
| Officer Omar Zidan  |              100.00 |
+---------------------+---------------------+
4 rows in set (0.01 sec)
```

Appendix 8; aggregation

```
mysql> SELECT
    ->     DATE_FORMAT(tv.ViolationDate, '%Y-%m') AS Month,
    ->     SUM(tv.FineAmount) AS TotalFines
    -> FROM
    ->     TrafficViolation tv
    -> GROUP BY DATE_FORMAT(tv.ViolationDate, '%Y-%m')
    -> ORDER BY Month;
+---------+------------+
| Month   | TotalFines |
+---------+------------+
| 2024-08 |     200.00 |
| 2024-09 |     600.00 |
| 2024-10 |     250.00 |
| 2024-11 |     100.00 |
+---------+------------+
4 rows in set (0.01 sec)

mysql>
```

Appendix 9; monthly fines collected

```
mysql> SELECT
    ->     tv.Location,
    ->     COUNT(tv.ViolationID) AS TotalViolations,
    ->     SUM(tv.FineAmount) AS TotalFines
    -> FROM
    ->     TrafficViolation tv
    -> GROUP BY tv.Location
    -> ORDER BY TotalViolations DESC;
+--------------+-----------------+------------+
| Location     | TotalViolations | TotalFines |
+--------------+-----------------+------------+
| Kenyatta Ave |               2 |     750.00 |
| Langata Road |               1 |     100.00 |
| Ngong Road   |               1 |     100.00 |
| Likoni Road  |               1 |     200.00 |
+--------------+-----------------+------------+
4 rows in set (0.00 sec)

mysql>
```

Appendix 10; violations by location