Technical Manual

Table of Contents

Installation Guide

The installation files are currently stored in the directory of the University of Nottingham systems.

Directory Structure

To begin modifying the code, download the installation files. The main PHP files for the website are stored in the InstallationFiles directory to make it easier to access the files which are publicly viewable by the user. The police-db.sql file and the script.js files are also located there to easily establish a connection between the PHP and JavaScript files.

Configuration

The database-details.php and user-session.php files are located in the config directory as these are crucial for setting up the website. The details for the connection to the database is in the database-details.php file.

All other PHP files contain the following code to access the database information stored in database-details.php:

```php
require("./config/database-details.php");
```

The connection to the database is then set up with the following line of code which is also used in every PHP file that contains SQL queries:

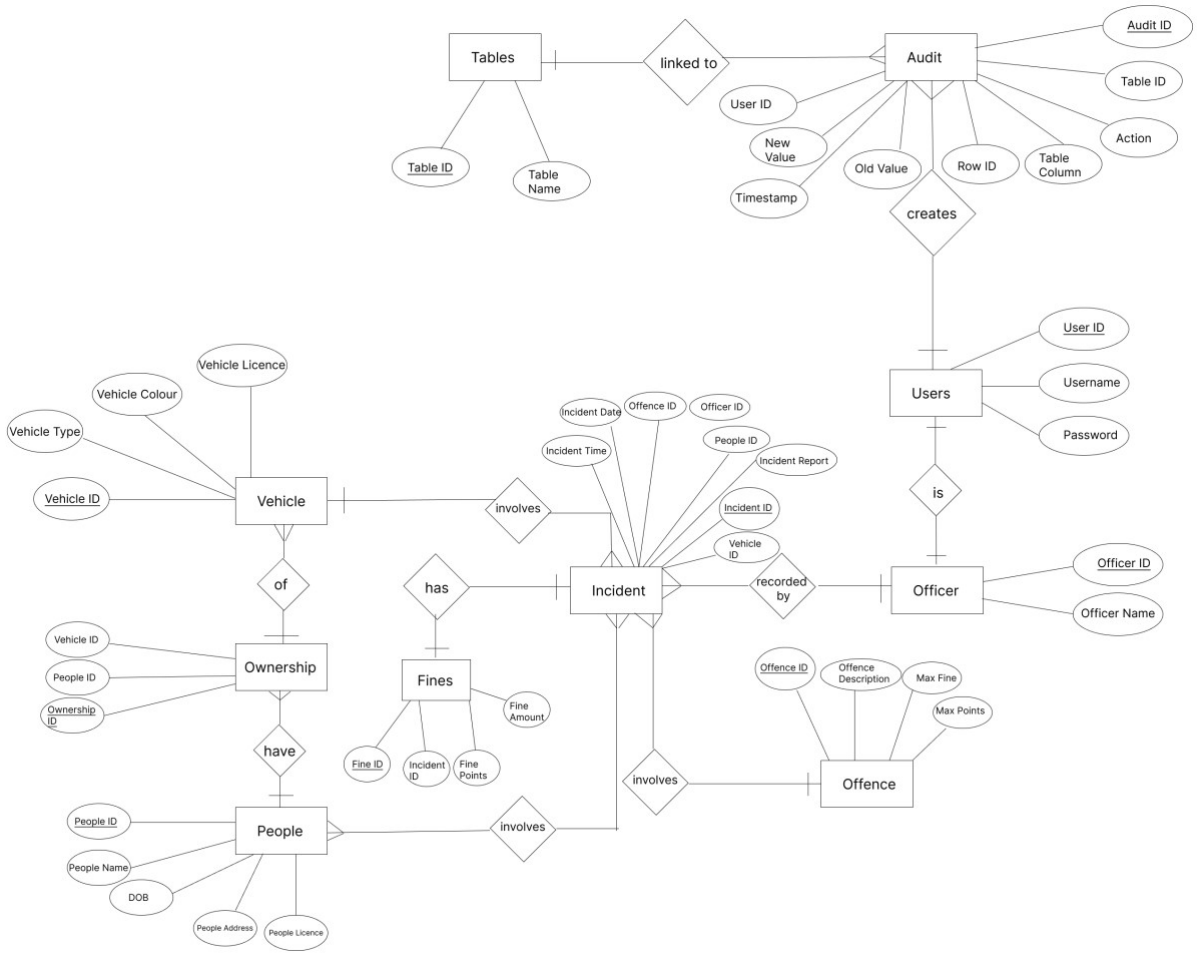
```php
// Open the database connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
```

The database can be accessed using a MySQL client such as MySQL Workbench. The SQL commands for constructing the tables for this database is in police-db.sql. To alter the database design through a MySQL client, download and import the police-db.sql file. Then modify or add new SQL commands to the PHP file. Some relevant example data has been used to fill most of the tables in the database.

Database Design

ERD

The police_db database contains 10 normalised tables that are based on the following entityrelationship diagram:



Relationships in ERD

The ERD above describes many relationships between the tables in the database. Vehicle and People has a many to one relationship because each vehicle has one owner but one person may own multiple vehicles. The Ownership entity facilitates this relationship by tracking Vehicle ID and People ID. It is assumed that each vehicle will only have a single owner so the Vehicle ID column has a unique constraint. However, the People ID column in the Ownership table can have duplicate entries because one person may own multiple vehicles. Hence, the People entity has a one to many relationship with the Ownership entity and the Vehicle entity has a many to one relationship with the Ownership entity.

The Incident entity has a many to many relationship with the People entity since an incident can involve multiple people and one person can be involved in multiple incidents. The Incident entity has a many to one relationship with the Vehicle entity. This is because the same vehicle can be mentioned in multiple incident reports but an incident report will only include one vehicle ID. Furthermore, the Incident entity has a one to one relationship with the Fines entity because it is assumed each incident is assigned only one fine.

Since many incidents can be recorded by one officer but only one officer is associated with each incident report, there is a many to one relationship between the Incident and Officer entities. A many to one relationship exists between the Incident and Offence entity because many incidents can involve the same offence but one offence is assigned to each incident report.

There is a one to one relationship between the Officer and Users entities because the Officer ID is a foreign key for the User ID which has a unique constraint. While not all users are officers, all officers are assigned a user ID, username, and password to allow them to log into the website.

The Users entity has a one to many relationship with the Audit entity because one user is responsible for producing multiple entries in the audit table through their interactions with the website. However, each entry in the audit table is only associated with a single user. The Audit entity is in a many to one relationship with the Tables entity because one table is linked to multiple audit table entries. However, one audit table entry only logs the changes enacted on a single table.

Audit Table Structure

Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	Default
Audit_ID	INT(11)	☑	☑	☐	☐	☐	☐	☑	
Table_ID	INT(11)	☐	☑	☐	☐	☐	☐	☐	
Action	VARCHAR(45)	☐	☑	☐	☐	☐	☐	☐	
Table_Column	VARCHAR(45)	☐	☐	☐	☐	☐	☐	☐	NULL
Row_ID	INT(11)	☐	☐	☐	☐	☐	☐	☐	NULL
Old_Value	VARCHAR(45)	☐	☐	☐	☐	☐	☐	☐	NULL
New_Value	VARCHAR(45)	☐	☐	☐	☐	☐	☐	☐	NULL
User_ID	INT(11)	☐	☑	☐	☐	☐	☐	☐	
Timestamp	DATETIME	☐	☑	☐	☐	☐	☐	☐	

The design for the Audit table was adapted from the advice in this article and this Stackexchange post. The goal of this Audit table is to enable the administrator to view everything that the user does using the frontend interface for the website. For instance, tracking the user's access to various records is useful from a legal and regulatory perspective. Hence, the Action column in the Audit Table will have the following values: READ, READ ALL, INSERT, or UPDATE.

The Table_Column, Old_Value, and New_Value columns have the VARCHAR(45) datatype because this Audit table is used to track all actions across the entire website. The datatype of the values inserted into these columns will, therefore, vary depending on which table is being audited.

The main key constraints that exist include the Primary Key constraint for Audit_ID which is also Not Null and auto increments. The Table_ID, Action, User_ID and Timestamp are all columns which can't have NULL values because this is the minimum amount of information that will be entered when a user views all the records for a table in the database. It is also important to track this information for legal purposes because knowing which user is accessing sensitive data from which table at what time is useful and important.

There are two foreign keys: Table_ID and User_ID. Table_ID references the primary key Table_ID in the table, Tables. Table_ID has On Update Cascade and On Delete Restrict set. These restrictions exist because making it possible to delete records of tables from the table, Tables, will create more ambiguity and potential confusion for the administrator auditing the website. On Update Cascade in both the Table_ID and User_ID foreign keys ensures that if there are changes to the IDs in the Tables and User tables, the Audit table will be updated accordingly and data integrity will be maintained.

User_ID references the Users table with On Update Cascade and On Delete No Action set. In addition to the above reasoning for setting User_ID to On Update Cascade, if the users' username is changed, that can be safely updated separately in the Users table without creating anomalies. However, if data for an user is deleted from the database (i.e. they no longer work in the police force), it will still be useful to have audit records associated with the deleted user. It will make it possible to distinguish those records from the records produced by active/current users of the website.

Fines Table Structure

Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI
Fine_ID	INT(11)	☑	☑	☐	☐	☐	☐	☑
Incident_ID	INT(11)	☐	☑	☐	☐	☐	☐	☐
Fine_Points	INT(11)	☐	☑	☐	☐	☐	☐	☐
Fine_Amount	INT(11)	☐	☑	☐	☐	☐	☐	☐

The Fines table has a primary key Fine_ID and all of the columns cannot have NULL values. This is because the fine is issued by the court then data about the fine is inserted into the Fines table. It is important that the administrator doesn't accidentally log an incomplete record for a fine by missing out crucial information and therefore no NULL values are allowed for any of the columns.

The Incident_ID is a foreign key that references the Incident_ID column in the Incident table with the following constraints: On Update Restrict and On Delete Restrict. This doesn't allow people to alter or delete the Incident_ID in order to prevent someone from tampering with records linked to fines. For instance, if a fine has been set for a certain incident, data integrity is preserved by ensuring that the Incident ID for that incident report can't be altered or deleted.

Incident Table Structure

Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI
Incident_ID	INT(11)	☑	☑	☐	☐	☐	☐	☑
Vehicle_ID	INT(11)	☐	☑	☐	☐	☐	☐	☐
People_ID	INT(11)	☐	☑	☐	☐	☐	☐	☐
Incident_Date	DATE	☐	☑	☐	☐	☐	☐	☐
Incident_Time	TIME	☐	☑	☐	☐	☐	☐	☐
Incident_Report	VARCHAR(500)	☐	☑	☐	☐	☐	☐	☐
Offence_ID	INT(11)	☐	☑	☐	☐	☐	☐	☐
Officer_ID	INT(11)	☐	☑	☐	☐	☐	☐	☐

The Incident Table allows for zero NULL values because each column corresponds with a key piece of information and it is important for officers to fill out the forms on the website. There are four foreign keys: Vehicle_ID, People_ID, Offence_ID, and Officer_ID. All of the foreign keys are set to On Update Cascade to prevent any problems if the primary keys of the referenced tables are altered. The People_ID, Vehicle_ID, and Offence_ID keys are also set to On Delete Restrict. This is because the incident reports will be rendered useless if data regarding the people, vehicle, or the offence is deleted from the tables that are referenced by the foreign keys in the Incident table. However, the Officer_ID foreign key is set to On Delete: No Action because police officers may be removed from the database when they are no longer a part of the police force. Since the Officer_ID primary key in the Officer table is set to auto-increment, the same Officer_ID will not be re-used and the Incident report will still be fairly useful.

Offence Table Structure

Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI
Offence_ID	INT(11)	☑	☑	☐	☐	☐	☐	☑
Offence_description	VARCHAR(50)	☐	☑	☐	☐	☐	☐	☐
Offence_maxFine	INT(11)	☐	☑	☐	☐	☐	☐	☐
Offence_maxPoints	INT(11)	☐	☑	☐	☐	☐	☐	☐

The Offence table exists to track offences based on the traffic guidelines and can easily be updated with new offences when laws change. The screenshot below lists the offences that are currently stored in the database and that can be cited in incident reports by police officers.

Offence_ID	Offence_description	Offence_maxFine	Offence_maxPoints
0	No offence	0	0
1	Speeding	1000	3
2	Speeding on a motorway	2500	6
3	Seat belt offence	500	0
4	Illegal parking	500	0
5	Drunk driving	10000	11
6	Driving without a licence	10000	0
8	Traffic light offences	1000	3
9	Cycling on pavement	500	0
10	Failure to have control of vehicle	1000	3
11	Dangerous driving	1000	11
12	Careless driving	5000	6
13	Dangerous cycling	2500	0

Table Structure: Officer

Column Name	Datatype	PK	NN
Officer_ID	INT(11)	☑	☑
Officer_name	VARCHAR(45)	☐	☑

Both columns in the Officer table cannot be null because the officer's name is entered into the database when a new user account is created/added. Hence, the Officer_ID is a foreign key that references the User_ID column in the Users table and is set to On Update Cascade and On Delete Cascade. Since all police officers are users, it is necessary to update and/or delete the Officer_ID if the User_ID is altered or deleted.

Table Structure: Ownership

Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI
Ownership_ID	INT(11)	☑	☑	☐	☐	☐	☐	☑
People_ID	INT(11)	☐	☑	☐	☐	☐	☐	☐
Vehicle_ID	INT(11)	☐	☑	☑	☐	☐	☐	☐
		☐	☐	☐	☐	☐	☐	☐

The Ownership table has 3 columns that have a Not Null constraint because any record entered into the ownership table will be incomplete if the people or vehicle information is missing. The Vehicle_ID has a unique index because a vehicle is assumed not to have multiple owners and therefore the same Vehicle_ID cannot be entered into the Ownership table multiple times. However, one individual can have multiple vehicles so there is no unique constraint on the People_ID column.

People_ID and Vehicle_ID are also foreign keys with On Update Cascade and On Delete Cascade set. This is because if vehicle or people records are updated, the changes will have to be reflected in the Ownership table to maintain data integrity. If the vehicle or person record is deleted completely from their respective tables, the corresponding information in the Ownership table should be deleted as well since it is no longer accurate if the vehicle or person doesn't exist.

Table Structure: People

Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	Default
People_ID	INT(11)	☑	☑	☐	☐	☐	☐	☑	
People_name	VARCHAR(50)	☐	☑	☐	☐	☐	☐	☐	
DOB	DATE	☐	☐	☐	☐	☐	☐	☐	NULL
People_address	VARCHAR(50)	☐	☐	☐	☐	☐	☐	☐	NULL
People_licence	VARCHAR(16)	☐	☑	☑	☐	☐	☐	☐	
		☐	☐	☐	☐	☐	☐	☐	

The People_licence column has the VARCHAR(16) datatype because drivers' licences are 16 characters in length. Driver's licences are also unique and therefore, the unique constraint ensures that it isn't possible to insert the same licence multiple times for multiple people. People_address and DOB are the only columns which have the default values of NULL because this information is comparatively less important for tracking individuals.

Table Structure: Tables

Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI
Table_ID	INT(11)	☑	☑	☐	☐	☐	☐	☑
Table_Name	VARCHAR(45)	☐	☑	☐	☐	☐	☐	☐

The Tables table was created for auditing purposes. Every time a new table is created, a new record is inserted into the Tables table.

Table Structure: Users

Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI
User_ID	INT(11)	☑	☑	☐	☐	☐	☐	☑
Username	VARCHAR(45)	☐	☑	☑	☐	☐	☐	☐
Password	VARCHAR(45)	☐	☑	☐	☐	☐	☐	☐

The Users table was created to track the account information for each user and facilitate logging users in and out of the website. It also makes it possible to restrict access to certain parts of the website to administrators only. The Username column has a unique constraint to prevent people from creating accounts with the same username. If multiple users have the same username, it makes it more difficult to differentiate between users – particularly when the administrator filters the audit table records for actions committed by specific users.

Table Structure: Vehicle

Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI
Vehicle_ID	INT(11)	☑	☑	☐	☐	☐	☐	☑
Vehicle_type	VARCHAR(20)	☐	☑	☐	☐	☐	☐	☐
Vehicle_colour	VARCHAR(20)	☐	☑	☐	☐	☐	☐	☐
Vehicle_licence	VARCHAR(7)	☐	☑	☑	☐	☐	☐	☐

This table has zero default values and all columns are not null because the type, colour, and licence plate number are all crucial pieces of information to record about a vehicle. Allowing for Null values could enables officers to skip entering certain data by accident and thereby create incomplete records.

The Vehicle_licence column has a unique constraint because each vehicle plate number is unique and therefore, it's possible to search for vehicles by their plate number on the website. The character length for the vehicle's type and colour are limited to 20 characters since this is a reasonable length for such data. The vehicle plate number is always 7 characters and therefore the datatype for this column is VARCHAR(7).

PHP Files: Configuration

user-session.php

I used the PHP code provided in this article to structure this file. user-session.php is stored in the config directory since it tracks user logins by checking whether the session variable for user_id is empty. If users haven't logged in but are attempting to access any part of the police traffic database website, they will instantly be redirected to the login page with the following code:

```
if (!isset($_SESSION['user_id'])) {
    header('location: login.php');
}
```

The time zone is also set and this enables insertion of date and time for each user action into the audit table.

```
date_default_timezone_set('Europe/London');
```

The following PHP code ensures that when the user clicks on the logout link in the navigation bar, the session will end and the session variables are unset or destroyed. The user is redirected to login.php.

```php
if (isset($_GET['logout'])) {
    session_destroy();
    unset($_SESSION['user_id']);
    unset($_SESSION['username']);
    unset($_SESSION['success']);
    header("location: login.php");
}
?>
```

nav-bar.php

The navigation bar contains links to all the PHP pages on the website. To prevent police officers from accessing pages restricted to admin, there is code that checks the username stored in the session variable 'username.' If this username is 'daniels', the following pages will be visible to them: 'Create Account,' 'Add Fine' and 'Audit Trail.'

```php
<?php
    if ($_SESSION['username'] == 'daniels') {
        echo '<li> <a href="#"> Admin </a>';
            echo '<ul>';
                    echo '<li> <a href="create-account.php"> Create
                    Account </a> </li>';
                    echo '<li> <a href="add-fine.php"> Add Fine </a> </
                    li>';
                    echo '<li> <a href="audit.php"> Audit Trail </a> </
                    li>';
            echo '</ul>';
        echo '</li>';
    }
?>
```

PHP Files: General Structure

At the top of each PHP file is code for tracking user login and logout via the session variables stored in user-session.php:

```php
require('./config/user-session.php');
```

Next, there is the HTML head tag followed by code for adding the navigation bar:

```php
require('nav-bar.php');
```

Most of the PHP pages will typically contain a form followed by code that ensures all errors are displayed, gets the database details from database-details.php and opens the database connection using these details.

```
error_reporting(E_ALL);
ini_set('display_errors',1);

require("./config/database-details.php");

// Open the database connection
$conn = mysqli_connect($servername, $username, $password, $dbname);

if(!$conn) {
    die ("Connection failed");
}
```

This is usually followed by an if block that checks whether form data has been POSTed by using isset(). For example:

```
if (isset($_POST['current_password'], $_POST['new_password'], $_POST
['confirm_password'])) {
```

The form data is accessed via the $_POST variables and used to construct a set of SQL queries. If these SQL queries are executed correctly, the website will inform the user that the task was successfully completed. There is also an else block which will be executed when there is an error and it provides the user with feedback. For example:

```
echo "Your username or password is incorrect. Please re-enter your
login details.";
```

There are also SQL queries for auditing the actions of the user on that web page and these queries insert new records into the Audit table.

At the bottom of each PHP file, the connection to the database is closed:

```
mysqli_close($conn);
```

Login PHP File

The login.php file works in tandem with the user-session.php file. The user must enter their username and password into the HTML form constructed below. Both form fields are required and the input type is text.

```
<form method="POST">
    Username: <input type="text" name="username" required><br/>
    Password: <input type="text" name="password" required><br/>
    <input type="submit" value="Log in">
</form>
```

When the user clicks the log in button, the session is started. To check that the user entered the correct username and password into the login form, the following query is executed. Every row in the Users table is selected where the username and password match the form data.

```
$sql = "SELECT * FROM Users WHERE Username = '$username' AND Password =
'$password'" ;
```

The of the SQL statement above should be a single row because each username is unique and therefore not repeated in the Username column. If the $result variable is a single row, the result array is looped through and the session variables are set using the User_ID and Username from the database.

```
if (mysqli_num_rows($result) == 1) {
    // PHP code for logging user in is adapted from: https://www.
    geeksforgeeks.org/how-to-display-logged-in-user-information-in-php/ -

    while($row = mysqli_fetch_assoc($result)) {
        // Set session variables
        $_SESSION['user_id'] = $row["User_ID"];
        $_SESSION['username'] = $row["Username"];
        $_SESSION['success'] = "You have logged in!";
    }
```

The user is logged in and redirected to index.php which is the home page for the police traffic website.

```
header('location: '.$redirect_url);
```

The index.php page introduces the various capabilities of the traffic website to new users.

Account PHP File

The account.php page enables the user to change their password by entering their current password and new password into an HTML form. All three form fields are required.

```
<form method="POST">
    Current Password: <input type="text" name="current_password"
    required><br/>
    New Password: <input type="text" name="new_password" required><br/>
    Confirm New Password: <input type="text" name="confirm_password"
    required>
    <input type="submit" value = "Update Password">
</form>
```

The first SQL query searches for the User whose username is stored in the session variable.

```
$sql = "SELECT * FROM Users WHERE Username = '$user'";
```

Then, the result of the above query is used to confirm that the current password entered by the user is correct. It also checks that the new password and the confirm new password fields match.

The update SQL query is executed next, thereby updating the user's password with the value stored in $new_password. This update is constrained with a WHERE clause which ensures that only the row where the Username is equal to the username of the current user is updated.

```
$update_sql = "UPDATE Users SET Password='$new_password' WHERE
Username='$user'";
```

Add Fine PHP File

The form for adding a fine is structured as follows:

```
<form method="POST">
    Report ID: <input type="number" min="0" name="incident_id" required>
    Penalty Points: <input type="number" name="fine_points" min="0"
    max='12' class='long_field' required>
    Fine Amount: <input type="text" name="fine_amount" pattern='[0-9]+'
    title='Only numbers' required>
    <input type="submit" value="Submit">
</form>
```

This form has a Report ID field with the type set to number and the minimum is set to zero to prevent users from entering negative numbers. The penalty points field has a maximum value of 12 since that is the maximum number of penalty points an individual can have on their driver's licence. The fine amount field is a larger number (i.e. 300) and so the input type for the form field is text to make it easier to enter the number. However, the pattern restricts acceptable inputs to integers 0 to 9 and the title 'Only numbers' provides user feedback if they enter anything that isn't an integer.

The SQL insert statement below inserts a new fine into the Fines table.

```
$fine_query = "INSERT INTO Fines (Incident_ID, Fine_Points,
Fine_Amount) VALUES ('$incident_id', '$fine_points', '$fine_amount')";
```

Audit PHP File

The audit file contains the following form which enables the administrator to filter the audit table by username, action, and/or table name. It is only necessary to fill out a single field of the form and click on the search button to view the relevant records.

```
<form method="POST">
    Username: <input type='text' name="user_name">
    Action: <input type='text' name="action" pattern='[a-zA-Z]+'
    title='Letters only'>
    Table: <input type='text' name="table_name" pattern='[a-zA-Z]+'
    title='Letters only'>
    <input type="submit" value = "Search" class='btn'>
    <input type="submit" name="view_all" value="View All" class='btn'>
</form>
```

The following SQL command is executed if the user clicks on the view all button and all the records are displayed below the search form.

```
if (isset($_POST['view_all'])) {
    $sql_search_all = "SELECT Audit.Audit_ID, Audit.Action, Audit.
    Table_Column, Audit.Row_ID, Audit.Old_Value, Audit.New_Value, Audit.
    Timestamp, Tables.Table_Name, Users.Username FROM Audit LEFT JOIN
    Tables ON Audit.Table_ID = Tables.Table_ID LEFT JOIN Users ON Audit.
    User_ID = Users.User_ID";
    $result_all = mysqli_query($conn, $sql_search_all);
```

If the user filters the records in the audit table by entering values into the form, the SQL query is tailored to append this data to the WHERE clause. For example, if they search for a specific username, the SQL will return all records where the Username column matches this username.

```
$search_audit_sql = "SELECT Audit.Audit_ID, Audit.Action, Audit.
Table_Column, Audit.Row_ID, Audit.Old_Value, Audit.New_Value, Audit.
Timestamp, Tables.Table_Name, Users.Username FROM Audit, Tables,
Users WHERE Audit.Table_ID = Tables.Table_ID AND Audit.User_ID =
Users.User_ID";

if ($username != '') {
    $search_audit_sql = $search_audit_sql." AND Users.Username =
    '$username'";
}

if ($action != '') {
    $search_audit_sql = $search_audit_sql." AND Audit.Action =
    '$action'";
}

if ($table != '') {
    $search_audit_sql = $search_audit_sql." AND Tables.Table_Name =
    '$table'";
}
```

Create Account PHP File

Here is the form for creating an account:

```
<form method="POST">
    Full Name: <input type="text" name="name" required><br/>
    Username: <input type="text" name="username" required><br/>
    Password: <input type="text" name="password" required><br/>
    Confirm Password: <input type="text" name="confirm_password" required>
    <input type="submit" value="Create Account">
</form>
```

An SQL select query is used to check if the username that was entered into the form is unique and not already present in the database:

```
$users_sql = "SELECT * FROM Users WHERE Username='$user'";
```

If zero rows are returned for the query above, the new user record is inserted into the Users table with the chosen username and password.

```
$new_user_query = "INSERT INTO Users (Username, Password) VALUES
('$user', '$password')";
```

Then, the User ID for the newly inserted user is selected and that User ID is assigned to the officer ID variable. This is because the Officer ID is a foreign key that references the User ID.

```
$get_id_query = "SELECT User_ID FROM Users WHERE Username =
'$user'";
$get_id_result = mysqli_query($conn, $get_id_query);
while($row = mysqli_fetch_assoc($get_id_result)) {
    $officer_id = $row['User_ID'];
}
```

A new record is inserted into the Officers table with the Officer_ID that corresponds to the User_ID from the Users table.

```
$new_officer_query = "INSERT INTO Officer (Officer_ID,
Officer_name) VALUES ('$officer_id', '$name')";
```

Edit Report PHP File

I referenced this article when structuring the PHP code for the edit-report.php page. This page automatically fills each field of the form with data from the database and then updates this data when the user makes changes.

This SQL select query was used to access all the relevant form data for the report_id:

```
$sql_search = "SELECT Incident.Incident_ID, Vehicle.Vehicle_licence, Vehicle.
Vehicle_ID, People.People_ID, People.People_name, People.People_licence, Incident.
Incident_Date, Incident.Incident_Time, Incident.Incident_Report, Offence.
Offence_description, Offence.Offence_ID FROM Incident, Vehicle, People, Offence
WHERE Incident.Vehicle_ID = Vehicle.Vehicle_ID AND Incident.People_ID = People.
People_ID AND Incident.Offence_ID = Offence.Offence_ID AND Incident.Incident_ID =
'$report_id'";
```

The report_id is appended to the URL when the user clicks on the 'Edit' link on the searchreports.php page and accessed using the following code:

```
$report_id = $_GET['report_id'];
```

The following SQL queries are used to set up options that the user can select from in the dropdown menus in the form. This also constrains the user by ensuring they can only submit valid inputs such as the name and driver licence for a person who is already in the database.

```
// Get offences from the database for dropdown
$offences_sql_search = "SELECT Offence_description FROM Offence";
$offences_result = mysqli_query($conn, $offences_sql_search);

// Get the list of people names and driver licences for dropdown
$name_licence_sql = "SELECT People_name, People_licence FROM People";
$name_licence_result = mysqli_query($conn, $name_licence_sql);

// Get the list of vehicle licence plates for dropdown
$vehicles_sql = "SELECT Vehicle_licence FROM Vehicle";
$vehicles_result = mysqli_query($conn, $vehicles_sql);
```

At the top of the PHP page, there are links where the officer can submit records for a new person or vehicle if they can't find that individual/vehicle in the drop down menus.

```
<p> Can't find the person or vehicle plate number you're searching for?
Add them to the database:
<p> <a href="submit-person.php"> Submit record for a new person</a></p>
<p> <a href="submit-vehicle.php"> Submit record for a new vehicle</a></p>
```

The following form field is hidden and therefore not shown to the user. It allows us to check whether the user has submitted the form by setting the value of 'update' to 1 when the form is submitted.

```
<input type="hidden" name="update" value="1" />
```

The value attribute is set equal to the PHP code that retrieves the incident date associated with this incident record.

```
Date: <input type="date" name="date" value="<?php echo $row
['Incident_Date']?>" required>
```

The searchable dropdown list was constructed using Selectize, a JS library, and code snippets taken from this StackOverflow post. No data is repeated in the dropdown because the values retrieved from the database for name and licence are compared to the value currently in the form field.

```
Offender:
<div class="select_box">
    <?php
    // Offender name and drivers licence dropdown list
    echo '<select name="name_and_licence" required>';
        echo'<option>'.$row['People_name'].' '.$row
        ['People_licence'].'</option>';
        while($name_licence_row = mysqli_fetch_assoc
        ($name_licence_result)) {
            if ($name_licence_row["People_name"] != $row
            ["People_name"] AND $name_licence_row["People_licence"]
            != $row["People_licence"]) {
                echo'<option>'.$name_licence_row["People_name"].' '.
                $name_licence_row["People_licence"].'</option>';
            }
        }
    echo '</select>';
    ?>
</div>
```

If the user clicks on the update button, the following query is submitted. There are SELECT queries nested within the main SQL update command because the Incident table contains multiple foreign keys (i.e. Vehicle ID, People ID, Offence ID).

```
$sql_update = "UPDATE Incident
SET Vehicle_ID = (SELECT Vehicle_ID FROM Vehicle WHERE Vehicle_licence =
'$plate_no'),
People_ID = (SELECT People_ID From People WHERE People_name =
'$driver_name' AND People_licence = '$driver_licence'),
Incident_Date = '$date',
Incident_Time = '$time',
Incident_Report = '$description',
Offence_ID = (SELECT Offence_ID FROM Offence WHERE Offence_description =
'$offence')
WHERE Incident_ID = '$report_id'";
```

Search Records PHP Files

The search-people.php, search-reports.php, and search-vehicles.php files contain PHP code and SQL queries which are structured similarly. The main difference between these files is the complexity of the SQL queries since search-vehicles.php uses data entered into a single field of form.

For example, the search-vehicles.php has a single WHERE clause that compares the vehicle licence to the plate number entered into the form. However, there are multiple LEFT JOINs because the website displays ownership information such as the person's name and driver's licence alongside the vehicle information.

```
$sql = "SELECT Vehicle.Vehicle_ID, Vehicle.Vehicle_type, Vehicle.
Vehicle_colour, Vehicle.Vehicle_licence, People.People_name, People.
People_licence, People.People_ID, Ownership.Ownership_ID FROM
Vehicle LEFT JOIN Ownership ON Vehicle.Vehicle_ID = Ownership.
Vehicle_ID LEFT JOIN People ON Ownership.People_ID = People.
People_ID WHERE Vehicle.Vehicle_licence LIKE '%$plate_no%'";
```

The search-reports.php file has code which is structured similarly to the audit.php file. Most importantly, values from additional form fields that are submitted are appended to the original SQL query via separate if statements. For example:

```
if ($report_id != '') {
    $sql_search = $sql_search." AND Incident.Incident_ID =
    '$report_id'";
}
```

Submit Records PHP Files

The submit-report.php, submit-person.php and submit-vehicle.php files are also all structured similarly. Submitted form data is inserted into the database with an SQL insert statement that accesses the values stored in the POST variables. For example:

```
$report_sql = "INSERT INTO Incident (Vehicle_ID, People_ID,
Incident_Date, Incident_Time, Incident_Report, Offence_ID,
Officer_ID) VALUES ((SELECT Vehicle_ID FROM Vehicle WHERE
Vehicle_licence = '$plate_no'), (SELECT People_ID FROM People WHERE
People_name = '$driver_name' AND People_licence =
'$driver_licence'), '$date', '$time', '$description', (SELECT
Offence_ID FROM Offence WHERE Offence_description = '$offence'),
$officer_id)";
```

Auditing

The SQL queries for tracking user interactions with the frontend website are insert statements and these insert statements are executed per user interaction. However, if a user chooses to view all the records – i.e. by clicking the view all button on the search reports page – 'READ ALL' will be inserted into the Action column of the Audit table. This also makes the Audit table easier to search later.

To add a new record to the Audit table, it is necessary to get the user_id which is stored in the session variable and the most current date and time info using the PHP getdate() function:

```
$user_id = $_SESSION['user_id'];
 $date=getdate(date("U"));
 $date_time = "$date[year]-$date[mon]-$date[mday] $date
 [hours]:$date[minutes]:$date[seconds]";
```

Example of an SQL insert query for the Audit table:

```
$audit_sql = "INSERT INTO Audit (Table_ID, Action,
Table_Column, Row_ID, Old_Value, New_Value, User_ID,
Timestamp) VALUES ((SELECT Table_ID FROM Tables WHERE
Table_Name = 'Users'), 'UPDATE', 'Password', '$user_id',
'$old_value', '$new_value', '$user_id', '$date_time')";
```

JavaScript and CSS Files

JavaScript

The script.js file contains the following function which implements a select box with a search option that makes the dropdown menus in my forms more efficient:

```
$(document).ready(function () {
    $('select').selectize({
        sortField: 'text'
    });
});
```

The JS Library used to implement this select box is JQuery and the code snippet above was taken from this Stackoverflow post.

CSS

The style.css file is located in the resources folder and the stylesheet is adapted from this Github repository. I have added a few styles at the bottom to modify the appearance of certain form elements. For example, the .btn is displayed inline-block which allows two buttons to be displayed side by side in a form (i.e. search and view all). The select box width is also altered

to make it longer in length which ensures that the width of all form fields remains consistent. Similarly, the long_field class is used to increase the width of shorter form fields.

```
.btn {
    display: inline-block;
}

.select_box {
    width: 250px;
}

.long_field {
    min-width: 25%;
}
```

The big_table class is used to set the maximum width on the table to ensure that the web page won't go on indefinitely if there are many records stored in the table. This is primarily useful when the user is searching the database and many results are returned or they wish to view all records stored in the database.

```
.big_table {
    display: block;
    height: 350px;
    overflow-y: scroll;
}
```